Modeling Reactive Game Agents using Cellular Automata in the Cell-DEVS Modeling Formalism

Alvi Jawad
Systems and Computer Engineering Department
Carleton University, Ottawa, Ontario
alvi.jawad@carleton.ca

Abstract— **Intelligent game agents are a vital part of modern games as they add life, story, and immersion to the game environment. The gaming industry's propensity toward realism has made the quest for more intelligent agents more important than ever before. Modeling and simulation of game agents and their surrounding environment provide an alternate setting to study dynamic agent behavior. Cellular Automata and the associated Cell-DEVS paradigm allows one to model such behaviors using the rigorously formalized Discrete Event Systems Specification (DEVS) formalism. In this project, we aim to model reactive game agents using the Cell-DEVS formalism and the CD++ toolkit. To analyze the dynamic behavior of many such agents, we perform a set of 13 experiments in widely varying system configurations. Our experimental results confirm the versatility of Cell-DEVS and the extended functionalities in the CD++ toolkit to model comfort-driven, exploratory, and desire-driven game agents and allow us to study fascinating reactive behavior resulting from drastic environmental changes.**

Index Terms—**Modeling & Simulation, Reactive Game Agents, Cellular Automata (CA), DEVS, Cell-DEVS**

I. Introduction

Game agents are indispensable entities that allow games to add lifelike immersion inside a virtual environment. Intelligent agents are the fundamental building blocks of any game, as the level of intelligent behavior determines the amount of realism the game can provide to the player. Many games, particularly those that make extensive use of autonomous agents, have grown increasingly complex over time. Consequently, the need to improve the agent behavior to match the entities that they mimic from the real-world has become a necessity. While many games have aged well over the years, many others have lost their popularity and player base due to not keeping up with this increased demand in realism.

The virtual nature of game environments turns actual experiments to study agent behavior into an impossible task. This, in turn, makes modeling and simulation a prime candidate to experiment on and improve such behavior. Game agents have been modeled using many different mathematical formalism, including cellular automata (CA) and influence maps [1], and finite automata [2], among others. The use of mathematical models allows the use formal methods to verify the correctness of the model and allow easy attunement of parameters to see widely varying agent behaviors. Additionally, this allows formally defined models to be easily translated to a different mathematical formalism and allows continuous improvements to the model.

A system-theoretic modeling formalism, named Discrete Event Systems Specification (DEVS), has garnered high research attention for modeling and simulation of artificial systems [3]. DEVS allows for creating single system components as modular atomic models and connecting multiple atomic and/or coupled models in a hierarchical manner to define complex models. DEVS has received many extensions over the past few years. One such extension is called Cell-DEVS that allows asynchronous modeling of timed cellular automata models [4]. The content-sharing and interoperability advantages of Cell-DEVS have proven that serious games can leverage the rigorous formalization and continuing standardization efforts on DEVS [5].

Cell-DEVS and the associated CD++ toolkit [6] have received their own extensions over the years. One such extension, added in [7], provides improved modeling capability by allowing each cell to make use of multiple state variables and neighbor ports. We aim to leverage this functionality to model intelligent game agents that react to changing game environments. Additionally, we wish to explore and analyze the dynamic behavior of reactive game agents under varying system configurations to gain insight into developing highly intelligent game agents.

The main contributions of this project are as follows.

- Definition of a cellular automata model of reactive game agents in the Cell-DEVS modeling formalism.
- Simulation of a multi-terrain game system and observation of the comfort-driven, exploratory, and desire-driven behavior of reactive game agents.
- Analysis of the conflicting interplay between desire and comfort for intelligent decision-making.

The rest of this report is structured as follows. Section II discusses the modeling formalism and concepts used for the remainder of the report. Section III introduces the core concept of reactive game agents and the modeled game terrain. Section IV provides a formal description of the reactive agents model. Section V details simulations performed using the developed model with comfort-driven agent behavior, whereas Section VI and Section VII describes simulations performed on agents with exploratory and desire-driven behavior, respectively. Section VIII discusses the numerous challenges faced during the project. Finally, Section IX concludes our work with a brief discussion of possible future extensions.

II. Background

In this section, we introduce the DEVS modeling formalism, the Cell-DEVS extension for modeling systems as CA, and some relevant terminologies and concepts used throughout this paper.

A. The DEVS Formalism

DEVS is a discrete-event modeling framework that allows the formal specification and modeling of systems that can be in discrete states at a time [8]. The state of the system changes based on either receiving input from the external world or internally after a certain amount of time has passed. Any system or system components that have a deterministic set of behavior can be modeled using DEVS. It provides an efficient way to build individual system components as modular atomic models and connecting many such atomic models through their interfaces to create coupled models. Coupled models can include two or more atomic, coupled, or a combination of the two to create highly hierarchical system models.

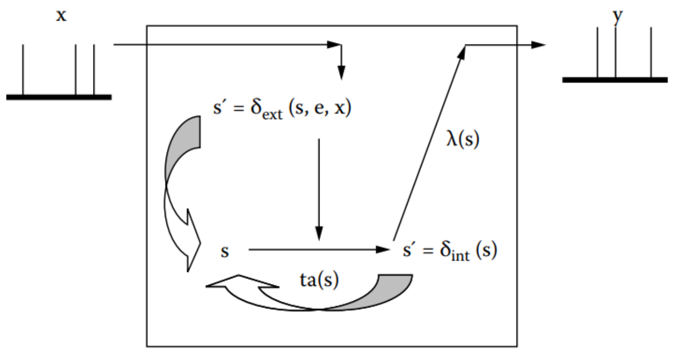


Fig. 1. DEVS atomic model semantics [8]

DEVS atomic models are highly modular, allowing them to be reused in any extension of DEVS through input and output ports. Figure 1 illustrates an informal depiction of how these atomic models function. At a time, the model can be one (s) of many discrete *states* (S). The *time advance function* ta is associated with the time a certain state is maintained, and after the continuous passage of ta, an internal transition will occur. Internal transitions are associated with internal events that can happen without the influence of external entities. A good example of this would be a generator model that may produce an output after a certain fixed period or following some stochastic distribution.

After an internal transition has occurred, the model triggers an *output function* λ that produces an *output* (y). This output is sent through an output port connected via an input port to one or more atomic and/or coupled models. Immediately after λ, an *internal transition function* δ_{int} performs a local state change, producing a *new state* (s') based on the current state, and modifies the time advance function to schedule the next internal event. This cycle of events continues until the time advance is set to infinity, a phenomenon we call the passivation, indicating that the model no longer produces any internal events.

Internal events, as well as passivated models, however, can be interrupted by inputs (x) arriving through the input port. The external input are handled by the *external transition function* δ_{ext}, that takes the input (x), the current state (s), and the elapsed time (e) since the last internal transition and produces a state change (s'). The time advance function is rescheduled to reflect the receipt and effect of external events and to prepare the system for the next internal event.

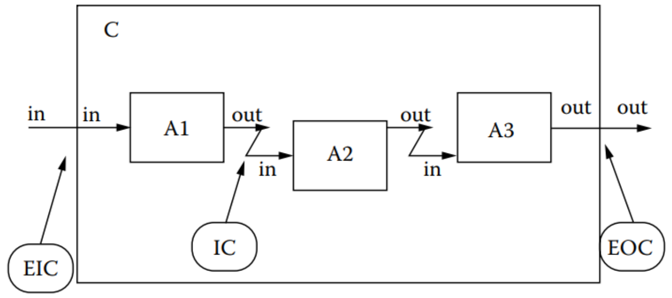


Fig. 2. DEVS coupled model semantics [8]

Outputs from an atomic model (A1) can be coupled with inputs to another atomic model (A2) using input-output ports to produce an *internal coupling* (IC), as shown in figure 2. Input from the coupled model directly (A1) or outputs to the coupled model directly (A3) are coupled using *external input* (EIC) and *external output* (EOC) *couplings*. These couplings link working atomic and/or coupled (A2 may very well be a coupled model) models and allow them to pass messages using their interfaces only. This approach allows us to build system models in a hierarchical manner, every module of which has high reusability due to their own set of actions and ability to connect to any other model through the abovementioned couplings.

To sum up, the DEVS modeling formalism allows one to build interoperable hierarchical models through the creation of modular atomic models and the ability to couple many such models. This essentially provides an abstract approach to separate modeling tasks from simulation and allows for independent verification and validation.

B. Modeling in Cell-DEVS

The Cell-DEVS paradigm, an extension of DEVS, allows defining systems as cellular models using the same semantics used in the DEVS formalism [4]. Each cell in the cell space is specified as a DEVS atomic model, as shown in figure 3. Atomic models take N inputs from all cells in a specified cell neighborhood through its interface. Inputs activate the local computing function (τ) that, after a predefined delay (d), produces a change in state (s'). This changed state may then be transmitted to other models through the cell interface. Delays belong to one of two types; *transport* and *inertial* delays. *Transport* delays model variable time delays and may schedule outputs in a queue, whereas in case of *inertial* delays, the output is preempted, meaning a newly computed value can overwrite all previously scheduled events in case of a state change.

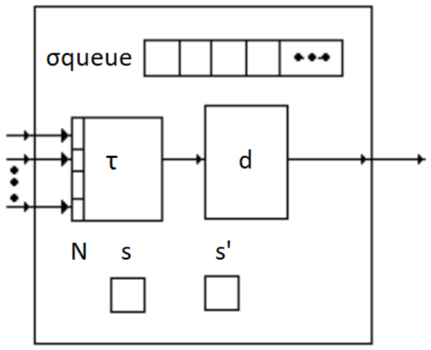


Fig. 3. Cell-DEVS atomic model semantics

Figure 4 shows a coupled Cell-DEVS model built after defining the connections between an atomic model and its neighbors. More than one atomic model can be defined to compose a coupled Cell-DEVS model, where each of the atomic models is connected to its neighborhood through input and output ports, similar to that of DEVS coupled models. The entire cell space is a $R \times C$ grid, where R denotes the number of rows, and C denotes the number of columns. The cell space can be either *wrapped* or *non-wrapped* (different behavior defined for border cells), resulting in a uniform and non-uniform neighborhood, respectively.

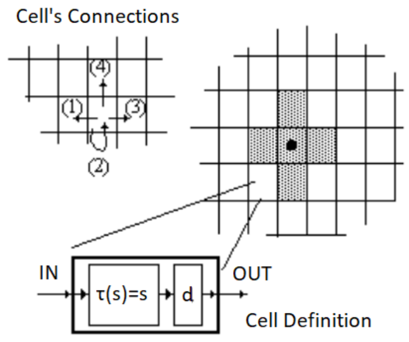


Fig. 4. Cell-DEVS coupled model

CD++ is a modeling and simulation environment based on the C++ programming language [6]. CD++ is capable of defining Cell-DEVS models using its own specification language and combining DEVS atomic models created in C++ to create hierarchical Cell-DEVS coupled models. An extension of the CD++ toolkit has added the capability to use multiple state variables for each cell as well as user-defined neighbor ports [7]. After the extension, the Cell-DEVS model specification now includes the size and the dimensions of the cell space, the shape of the neighborhood, the number and initial values of state variables, and the definition of neighbor ports (optional) alongside any rules for cell and border cell behavior. A cell's local computing function is defined using a set of rules in the following format:

$$\{postconditions\}\{assignations\}\{delay\}\{preconditions\}$$

Each rule for a cell must be specified in this format, with the exception of *assignations*. If the *postconditions* defined for a particular rule is satisfied, after the specified *delay* for that rule, *assignations* to state variables will be made (optional), and then the *postconditions* will be evaluated, usually resulting in one or more port assignations so the current state of the cell, as well as any output messages, can be sent via the defined neighbor ports. New rules are evaluated sequentially after the failure of the previous one. If no rules apply for a cell in the cell space, or more than one rule applies, an error will be produced indicating the conflicting behavior of cells. This simple syntax allows one to define simple as well as complex behavior of cells, giving them the flexibility to model extremely complicated hierarchical cellular models.

III. Reactive Game Agents

A. Reactive Agents

Agents are a crucial part of game worlds where they represent the individual non-player characters (NPCs) with their distinct personality and behavior that give the game life, story, and atmosphere. Agents can also be monsters or enemies that the player needs to fight or allies or soldiers in the player's army. Reactive game agents are flexible living entities that can react to changes in their surrounding environment. The core idea of this concept is derived from [1], where the authors combine influence maps with cellular automata to represent a 3D game world named EmerGEnT that model natural phenomena such as fluid flow, heat, fire, pressure, and explosions. The agents in their game world are able to react and move to other cells based on their comfort and desire levels in the current and neighboring cells with varied reaction times and exhibit different levels of intelligent behavior. In this section, we describe our game environment and the behavior of different agent types to be modeled and simulated in subsequent sections.

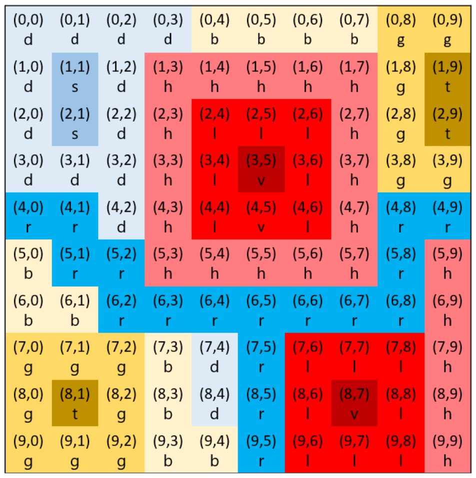
B. The Game Environment

The game environment is a 10x10 cell space where each cell represents a terrain of a certain type. Each cell hosts different levels of fire, wind, and water, resulting in a certain value of heat, pressure, and wetness for an agent if they were to stand atop that cell. Based on the values of heat, pressure, and wind, we divide the terrains into three different categories. These terrains are fire-attribute terrains (volcano, lava, heated), wind-attribute terrains (tornado, gust, breeze), and water-attribute terrains (river, storm, drizzle). Each of the three terrain types belonging to each category represents the primary level of element in that cell in descending rate. As an example, for fire-attribute cells, volcano cells have the highest level of heat, followed by reduced levels of heat in lava and heated cells.

TABLE I
CELL VALUES FOR DIFFERENT TERRAIN TYPES

Terrain	Symbol	Celltype	Heat	Pressure	Wetness
Fire Attribute Terrains					
Volcano	v	0.001	100	50	0
Lava	l	0.002	80	30	0
Heated	h	0.003	40	20	20
Wind Attribute Terrains					
Tornado	t	0.004	0	100	40
Gust	g	0.005	0	60	20
Breeze	b	0.006	0	30	10
Water Attribute Terrains					
River	r	0.007	0	20	100
Storm	s	0.008	20	60	70
Drizzle	d	0.009	0	10	40

The nine different terrain types and associated heat, pressure, and wind values are represented in table I. The values used for each terrain are fictional and are assigned to see different agent behavior in different terrains. All cells in the game environment belong to one of these nine terrains. Figure 5 is a corresponding figure that shows the game terrain with specific terrain types in different cell coordinates and different colors used for the visualization of our simulations. This figure is important as the same terrain will be used throughout the report to perform different simulations.



(0,0) d	(0,1) d	(0,2) d	(0,3) d	(0,4) b	(0,5) b	(0,6) b	(0,7) b	(0,8) g	(0,9) g
(1,0) d	(1,1) s	(1,2) d	(1,3) h	(1,4) h	(1,5) h	(1,6) h	(1,7) h	(1,8) g	(1,9) t
(2,0) d	(2,1) s	(2,2) d	(2,3) h	(2,4) l	(2,5) l	(2,6) l	(2,7) h	(2,8) g	(2,9) t
(3,0) d	(3,1) d	(3,2) d	(3,3) h	(3,4) l	(3,5) v	(3,6) l	(3,7) h	(3,8) g	(3,9) g
(4,0) r	(4,1) r	(4,2) d	(4,3) h	(4,4) l	(4,5) v	(4,6) l	(4,7) h	(4,8) r	(4,9) r
(5,0) b	(5,1) r	(5,2) r	(5,3) h	(5,4) h	(5,5) h	(5,6) h	(5,7) h	(5,8) r	(5,9) h
(6,0) b	(6,1) b	(6,2) r	(6,3) r	(6,4) r	(6,5) r	(6,6) r	(6,7) r	(6,8) r	(6,9) h
(7,0) g	(7,1) g	(7,2) g	(7,3) b	(7,4) d	(7,5) r	(7,6) l	(7,7) l	(7,8) l	(7,9) h
(8,0) g	(8,1) t	(8,2) g	(8,3) b	(8,4) d	(8,5) r	(8,6) l	(8,7) v	(8,8) l	(8,9) h
(9,0) g	(9,1) g	(9,2) g	(9,3) b	(9,4) b	(9,5) r	(9,6) l	(9,7) l	(9,8) l	(9,9) h

Fig. 5. The Game Terrain

C. *Comfort-Driven Agents*

The first agent type that we model are agents that put comfort before everything else. Agents feel a certain level of comfort in each cell based on what type of agent they are. For example, human agents may feel the most uncomfortable if they were to stand on a fire-attribute cell but deal well with wind and water-attribute cells. A fiery monster, on the other hand, may feel perfectly fine on a fire-attribute cell but might die instantly if forced to stand on a water-attribute cell. We assign weights to heat, pressure, and wetness that allow us to easily alter the agents' comfort preferences and model many different agent types.

Each agent, while standing atop a certain cell, determine their current level of comfort and the prospective comfort if they were to move to a certain cell. If they find a nearby cell that is more comfortable than their current cell, they move to that cell. Agents also have a right-moving tendency; if all cells are equally comfortable, agents always prefer moving to the right. Agents can move in all eight cardinal and ordinal directions and will move up, down, directly or diagonally left if directly or diagonally moving to the right are not possible. In our experiments, agents start anywhere in the terrain and try to cross the terrain from left to right without colliding with any other agent in the system. While having this right-moving behavior, they always consider their current level of comfort and may move in opposite directions or take turnarounds if necessary to be as comfortable as possible.

D. *Desire-Driven Agents*

In addition to these terrain types, each cell has certain properties that lead to that cell having certain desirability for an agent. Similar to the comfort case, agents also prefer moving to cells that have the highest desirability for them as long as the movement is not too uncomfortable. The desirability of cells may change after a certain time, and the agents should react accordingly to those changes. Based on the rules, the reactive agents can react to their environment based on comfort only (comfort-driven agents), desire only (desire-driven agents), or a combination of both. Our final objective is to observe the interplay between desire and comfort and see what happens when both are equally important or when one dominates the other.

IV. The Reactive Agents Model Definition

In this section, we provide the formal specification for the *ReactiveAgents* model, followed by modeling the system using CD++ and testing different simulation scenarios.

A. *Neighborhood Selection*

We select the extended Moore neighborhood to use in our model. This choice allows us to model the movement of the agent in all eight cardinal (North, South, East, West) and ordinal directions (Northeast, Northwest, Southeast, Southwest) after selecting the cell that has the highest comfort level (lowest cell value) for the agent. As shown in figure 6, the cell marked in green and yellow represents the agent and its immediate neighborhood, respectively, whereas the grey cells represent the extended neighborhood. This second level allows us to avoid conflict with any other agent that is trying to compete for the same cell from all straight and diagonal directions.

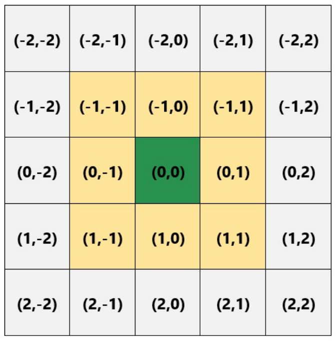


Fig. 6. The Cell Neighborhood

B. Formal Specification

The *ReactiveAgents* atomic model is formally specified as follows:
RA_Atomic =

$$< X, Y, S, N, type, d, \tau, \delta_{int},\ \delta_{ext},\ \lambda, ta >$$

- X = $\{\phi\}$;
- Y = $\{\phi\}$;
- S = (refer to section IV-C)
- N = {
  $(-2,-2), (-2,-1), (-2,0), (-2,1), (-2,2),$
  $(-1,-2), (-1,-1), (-1,0), (-1,1), (-1,2),$
  $(0,-2), (0,-1), (0,0), (0,1), (0,2),$
  $(1,-2), (1,-1), (1,0), (1,1), (1,2),$
  $(2,-2), (2,-1), (2,0), (2,1), (2,2)$
  };
- type = transport;
- d = 100; // in milliseconds
- $\tau : N \rightarrow S$; // (refer to section IV-E)
- $\delta_{int},\ \delta_{ext},\ \lambda$, and ta are defined using the Cell-DEVS specification

The *ReactiveAgents* coupled model is formally specified as follows:
RA_Coupled =

$$< Xlist, Ylist, X, Y, I, \eta, N, \{r, c\}, C, B, Z, select >$$

- Xlist = $\{\phi\}$;
- Ylist = $\{\phi\}$;
- X = Y = (refer to section IV-C)
- I = $< P_x, P_y >$
- η = 25;
- N= {
  $(-2,-2), (-2,-1), (-2,0), (-2,1), (-2,2),$
  $(-1,-2), (-1,-1), (-1,0), (-1,1), (-1,2),$
  $(0,-2), (0,-1), (0,0), (0,1), (0,2),$
  $(1,-2), (1,-1), (1,0), (1,1), (1,2),$
  $(2,-2), (2,-1), (2,0), (2,1), (2,2)$
  };
- C = $\{C_{ij}$ / $i \in [0,9],\ j \in [0,9]\}$;
- B = ϕ; // wrapped border
- Z = the translation function that determines the dynamics of the agent behavior // (refer to section IV-E)
- select = {
  $(0,0), (0,1), (-1,1), (1,1), (-1,0),$
  $(1,0), (-1,-1), (1,-1), (0,-1)$
  }

C. State Variables

The extended CD++ specification allows the use of multiple state variables in each cell [7]. We leverage this capability to have each cell host six different state variables, namely `agent`, `cell`, `heat`, `pressure`, `wetness`, and `desirability`. The description of each state variable and the values that each variable is allowed to have is as follows:

- `agent`: the type of agent in each cell
  - 0: No agent
  - 1: Human agent (afraid of fire)
  - 2: Fiery Monster agent (afraid of water)
  - 3: Vapor Alien agent (afraid of pressure)
- `cell`: The terrain type; belongs to one of the terrains as illustrated in table I.
  - {0.001, 0.002, 0.003, ... 0.009} based on the terrain
- `heat`: the amount of fire heat in each cell
  - {0, 1, 2, ... 100} integers from 0 to 100
- `pressure`: the amount of wind pressure in each cell
  - {0, 1, 2, ... 100} integers from 0 to 100
- `wetness`: the amount of water wetness in each cell
  - {0, 1, 2, ... 100} integers from 0 to 100
- `desirability`: the desirability of a cell to an agent
  - 0: No desirability
  - 0.99: Maximum desirability
  - all real numbers between 0 and 0.99: specific level of desirability

All cells start with certain levels of fire, wind, and rain on them, represented by integer values from 0-100 of the state variables `heat`, `pressure`, and `wetness` respectively. These values determine the level of comfort for each agent based on what weights are assigned to each of these attributes. The specific values assigned also determines the terrain type (refer to table I), reflected in the `cell` state variable. While we do not make use of this variable right now, this value will be used extensively when we model the explorer agent in section VI. Similarly, the state variable `desirability` will only be used when modeling our desire-driven agents in section VII.

An agent can only occupy one terrain space (one square cell space) at a time. In most cases, we model multiple instances of a single type of agent, in which case the 0 or 1 values of the state variable `agent` defines whether an agent is occupying a cell or not. When the simulation starts, all the positions of agents on the terrain are updated synchronously, and they either move, stay in the same cell or die based on our set of rules. We use a wrapped boundary and transport delays in our model.

D. Neighbor Ports

The extended CD++ specification allows the modeler to define their own output ports for each cell. We define four ports, namely `occ`, `celltype`, `comfort`, `desire`. The `occ` port, depending on the type of simulation being performed, outputs the value of either the `agent` variable, or a sum of the `agent` variable and another additional value. This is the primary port that will be used for the visualization of the simulations. The `celltype` and `desire` ports simply output the values assigned to the `cell` and `desirability` state variables in that cell. Conversely, the `comfort` port outputs the value after calculating the comfort of a certain agent using the `heat`, `pressure`, and `wetness` state variables and the assigned weights to each of them.

This calculation of the comfort of an agent is done using the macro COMFORT_LEVEL. The COMFORT_LEVEL of an agent is defined as:

```
CL=heat*W_H + pressure*W_P + wetness*W_W
```

`W_H, W_P,` and `W_W` are the weight of the discomfort felt by each agent by heat, pressure, and wetness, respectively. For example, in our experiments, we model three primary fictional agents, namely humans, fiery monsters, and vapor aliens. There are also four additional secondary agents that we do not name but model briefly to see the behavior in case of equal weights assigned to all or any two of the elements.

Humans feel most afraid of fire, but they do not care too much about a strong wind or getting wet. Therefore, `W_H, W_P,` and `W_W` are assigned the values 0.007, 0.002, and 0.001 respectively. For a monster agent that lives inside volcanoes and is primarily afraid of water, we instead assign the values 0.0, 0.001, 0.009, respectively. Similarly, a vapor alien is minimally affected by both fire and water but is tremendously impacted by wind, so we assign the values 0.0001, 0.0098, 0.0001, respectively, to see agent behavior in the extremities. These values can be altered easily to model the behavior of any agent that we want. Table 1 represents the different agents and the specific weights assigned to `W_H, W_P,` and `W_W` when simulating those agents.

TABLE II
TYPES OF AGENTS AND ASSOCIATED ELEMENT WEIGHTS

Agent ID	Agent Type	W_H	W_P	W_W
Primary Agents				
Agent01	Human	0.008	0.002	0.001
Agent02	Fiery Monster	0	0.001	0.009
Agent03	Vapor Alien	0.0001	0.0098	0.0001
Secondary Agents				
Agent04	N/A	0.005	0.005	0
Agent05	N/A	0	0.005	0.005
Agent06	N/A	0.005	0	0.005
Agent07	N/A	0.0033	0.0033	0.0033

E. Rules

We define nine sets of rules to define the reactive behavior of agents. The zeroth ruleset is called the Insta-death rule, as this rule dictates when agents die or turn to nothingness (zero). The next eight sets of rules determine the movement tendencies of an agent upon feeling uncomfortable. Finally, an additional default rule dictates the behavior of cells where no agent death or movement occurs. The rules are as follows:

1) The Insta-death Rule: This is the zeroth rule that allows us to model agent death. While standing on top of a terrain, depending on the terrain type and the type of agent, the agent can be in an extremely uncomfortable state. This state occurs when the macro COMFORT_LEVEL calculates discomfort to be 80% or above.

- Rule 0: (Death) If an agent has reached or exceeded the extreme discomfort level of 80%, that agent dies immediately.

A good example of this would be human agents on volcanoes where the COMFORT_LEVEL of volcano cells for human agents are calculated to be exactly 80%. As we will see in our simulations, whenever an agent is at or beyond this discomfort threshold of 80%, that agent will immediately die. This rule has the overall highest priority among all rules.

2) The Movement Rules: Rules 1-8 determines the agents' movement upon confirming the comfort values of neighboring cells. At each time step, an agent calculates the COMFORT_LEVEL of its current cell and all the neighboring cells in the Moore level 1 neighborhood (the immediate neighborhood) to determine the next most comfortable position. If there is a clear winner among all neighboring cells, the agent will move to that cell, becoming more comfortable than before. In case of a tie, agents always try to move toward the right, either directly, or diagonally, if possible. Direct movements are always preferred over diagonal movements, and agents only move upward, downward, or directly, or diagonally left if moving to the right is less comfortable.

Each of the Movement rules is comprised of two rules to model the leaving and arriving behavior of agents in each case. All of the movement rules are evaluated only in the case of agents that are not in a state of extreme discomfort. A brief description of all the movement rules are as follows:

- Rule 1: (Move East) If an agent is below the extreme discomfort threshold, and the east cell is empty and the most comfortable cell out of all the immediate neighbors, move to that cell. This rule has the highest priority among all movement rules and the second-highest priority among all rules.
- Rule 2: (Move Northeast) If an agent is below the extreme discomfort threshold, and the northeast cell is empty and the most comfortable cell out of all the immediate neighbors, and no other agent is trying to move to that cell using rule 1, move to that cell. This rule has the third-highest priority.
- Rule 3: (Move Northwest) If an agent is below the extreme discomfort threshold, and the southeast cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-2, move to that cell. This rule has the fourth-highest priority.

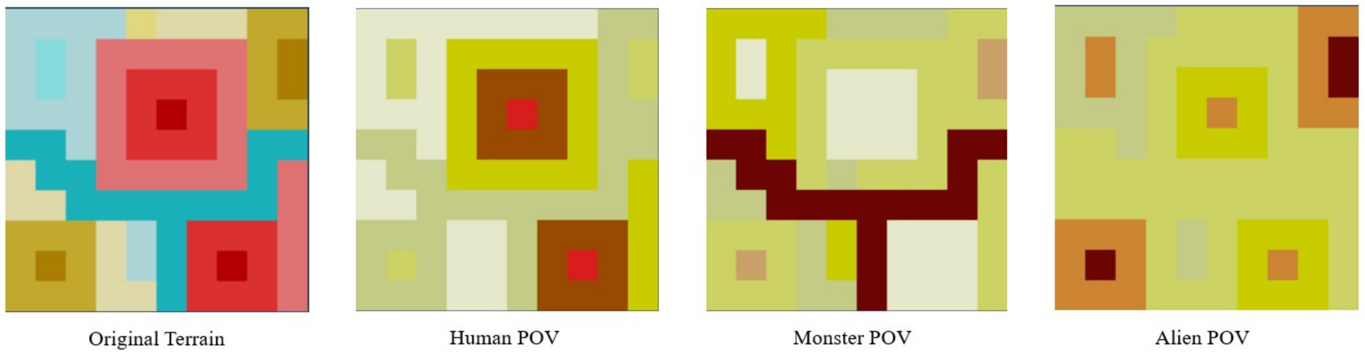


Fig. 7. The game terrain from the agents' point of view (POV)

- Rule 4: (Move North) If an agent is below the extreme discomfort threshold, and the north cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-3, move to that cell. This rule has the fifth-highest priority.
- Rule 5: (Move South) If an agent is below the extreme discomfort threshold, and the south cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-4, move to that cell. This rule has the sixth-highest priority.
- Rule 6: (Move Northwest) If an agent is below the extreme discomfort threshold, and the northwest cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-5, move to that cell. This rule has the seventh-highest priority.
- Rule 7: (Move Southwest) If an agent is below the extreme discomfort threshold, and the southwest cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-6, move to that cell. This rule has the eighth-highest priority.
- Rule 8: (Move West) If an agent is below the extreme discomfort threshold, and the west cell is empty and the most comfortable cell out of all the immediate neighbors and no other agent is trying to move to that cell using rules 1-7, move to that cell. This rule has the ninth-highest priority.

Overall, Rule 1 dictates a straight movement to the right. Following Rule 1, Rules 2 and 3 represent the agents' tendency to move diagonally to the right whenever a straight movement is not comfortable. Rules 4 and 5 represent the agents' tendency to move upward or downward whenever a straight or diagonal movement to the right is not comfortable. Rules 6 and 7 represent the agents' tendency to move diagonally left whenever a straight or diagonal movement to the right or an upward or downward movement is not comfortable. Finally, Rule 8 represents the agents' tendency to move to the left when no other directional movement is comfortable.

3) The Default Rule: This rule is evaluated when all other rules fail to satisfy and has the lowest priority among all rules.

- Default Rule: (Retain State) Regardless of the cell state, maintain the same state.

All of these rules make up the core behavior of our reactive agents and what they do as they wish to feel comfortable as well as move toward the right when they can comfortably do so. Additional rules will be added when we try to simulate multiple agents at the same time, as well as in the case of our explorer and desire-driven agents.

F. Files

The detailed implementation of the rules can be found and altered in the *reactiveagent.ma* file in individual folders inside the root project directory. We make extensive use of macros to reduce the rule size; the file *macro_comfort.inc* contains macros to calculate and output the COMFORT_LEVEL of each cell for a particular agent. This macro is extremely useful, as the assigned weights to each element can be easily changed to observe widely varying agent behaviors. The file *macro_min.inc* contains macros to calculate the minimum COMFORT_LEVEL of all nine neighboring cells, whereas, *macro_directions.inc* contains macros for easier representation of the eight possible movable directions. Finally, the file *macro_desirability.inc* contains macros to spread the desirability and how agents behave when both comfort and desirability becomes important. Until we start modeling desire-driven agents, these functionalities will not be used.

V. Simulating Reactive Agents

In this section, we simulate the reactive agents model with different agents and observe their behavior to verify the correctness of our rules as well as identify any interesting pattern.

A. Experimentation Strategy

Figure 7 shows the original game terrain and how it looks from our three primary agents' point of view (POV). The colors used in the original terrain match those illustrated in

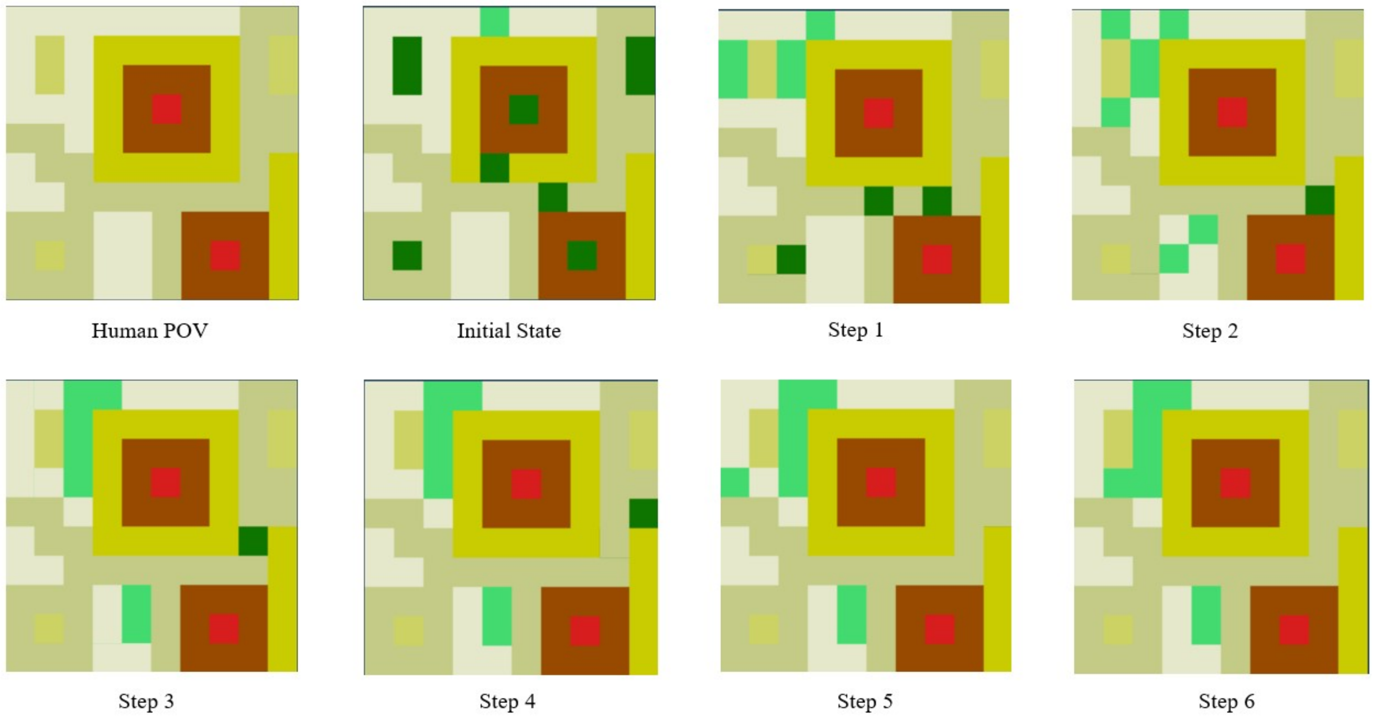


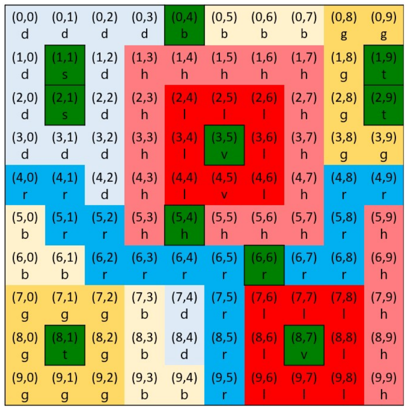
Fig. 8. Full simulation run with ten human agents

figure 5 to maintain consistency. Looking at the other three images, we can see that even for the same cell, the level of comfort felt by different agents is different from each other. We use three shades of red to represent high discomfort (above 70%), three shades of orange to represent moderate discomfort (between 40% and 70%), and finally three shades of yellow to represent low discomfort (between 10% and 40%). White cells represent minimal discomfort (below 10%), and an agent becomes comfortable upon moving to such a cell.

For all of our simulations, we will use ten agents of a certain type placed at predefined locations. Then we will simulate different scenarios by assigning different weights to different elements to simulate the behavior of seven different agents mentioned in table II. The core idea here is to see the spreading and flocking behavior of agents as each agent type is likely to behave differently than the others. Simulations will be stopped after either ten seconds or upon reaching a steady final state (agents moving back and forth), whichever is earlier. In each case, we will record videos that can be found in the results folder in separate folders named after the experiments and agents.

Simulations can be run in a Linux environment by following the guidelines provided in the *README.md* file inside the *comfort-driven_agents* folder of the root project directory. The experiments can be replicated by assigning the same weights and can be viewed using the *ra_comfort-driven.json* and the *reactiveagent.log01* files in the ARSLab Simulation viewer[1] hosted by GitHub. While the *.log01* files will be automatically created after running the simulations, we will provide the log files for each experiment in the same folder as the experiment videos for easier visualization.

[1]https://simulationeverywhere.github.io/CD-WebViewer-2.0/index.html



(0,0) d	(0,1) d	(0,2) d	(0,3) d	(0,4) b	(0,5) b	(0,6) b	(0,7) b	(0,8) g	(0,9) g
(1,0) d	(1,1) s	(1,2) d	(1,3) h	(1,4) h	(1,5) h	(1,6) h	(1,7) h	(1,8) g	(1,9) t
(2,0) d	(2,1) s	(2,2) d	(2,3) h	(2,4) l	(2,5) l	(2,6) l	(2,7) h	(2,8) g	(2,9) t
(3,0) d	(3,1) d	(3,2) d	(3,3) h	(3,4) l	(3,5) v	(3,6) l	(3,7) h	(3,8) g	(3,9) g
(4,0) r	(4,1) r	(4,2) d	(4,3) h	(4,4) l	(4,5) v	(4,6) l	(4,7) h	(4,8) r	(4,9) r
(5,0) b	(5,1) r	(5,2) r	(5,3) h	(5,4) h	(5,5) h	(5,6) h	(5,7) h	(5,8) r	(5,9) h
(6,0) b	(6,1) b	(6,2) r	(6,3) r	(6,4) r	(6,5) r	(6,6) r	(6,7) r	(6,8) r	(6,9) h
(7,0) g	(7,1) g	(7,2) g	(7,3) b	(7,4) d	(7,5) r	(7,6) l	(7,7) l	(7,8) l	(7,9) h
(8,0) g	(8,1) t	(8,2) g	(8,3) b	(8,4) d	(8,5) r	(8,6) l	(8,7) v	(8,8) l	(8,9) h
(9,0) g	(9,1) g	(9,2) g	(9,3) b	(9,4) b	(9,5) r	(9,6) l	(9,7) l	(9,8) l	(9,9) h

Fig. 9. Initial Agent positions

B. Simulating Single Comfort-driven Agents

Figure 9 illustrates the starting position of human agents in experiment 01. We use dark green to represent an agent in its uncomfortable state, and light green to represent its comfortable state. The initial locations are determined by filling the most extreme cells of each element type (e.g., volcano, tornado, and river cells) as we want to see agent deaths. This is followed by placing agents in less extreme positions in no particular preference. Any other experiment can

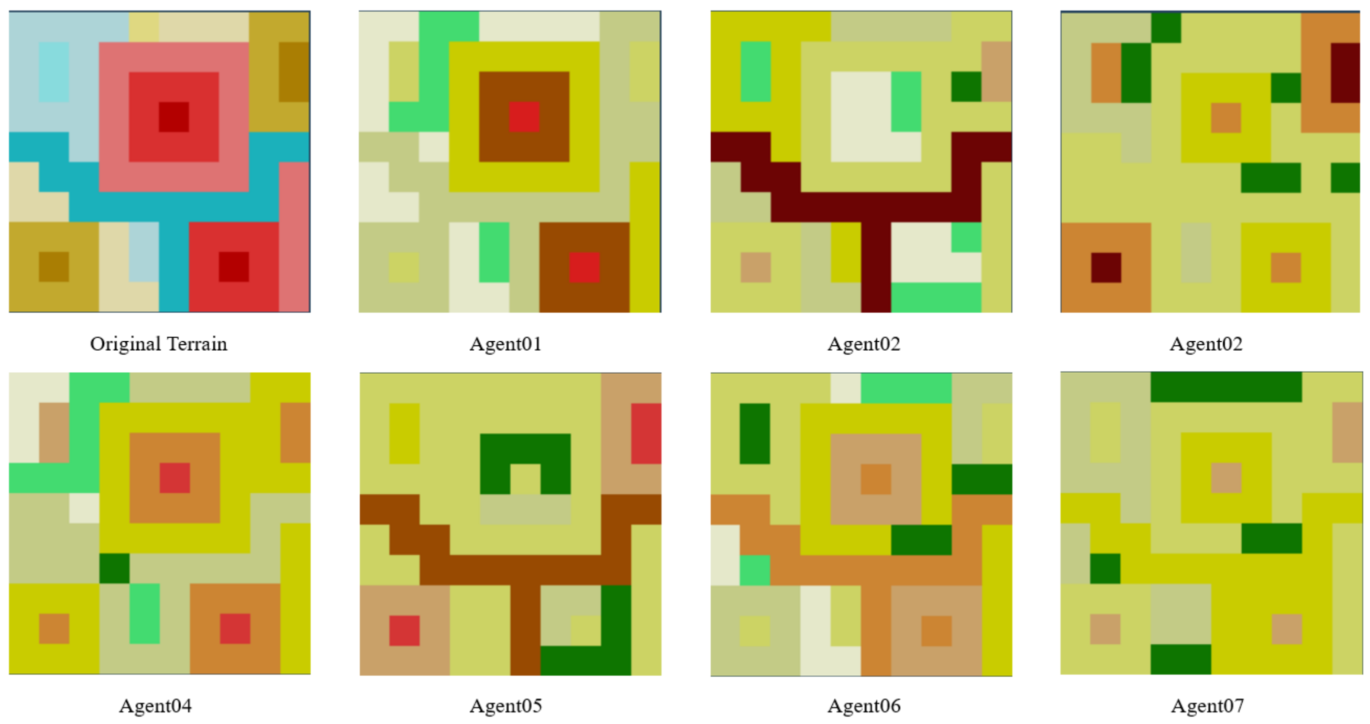


Fig. 10. Experiment 01: Final states from the POV of all seven agent types

be performed by changing the values of the *reactiveagent.val* file inside *comfort-driven_agents* folder.

Experiment 01: Figure 8 shows a full run of experiment 01 using ten human agents until the simulation reaches a steady state. We can see the state of the game terrain from a human agent's POV without and with agents in the first two snippets. In Step 1, the two agents previously standing on volcano terrains die immediately, and the number of agents goes down to eight. Simultaneously, agents in the middle go down to the river, and agents to the top-left move to areas with drizzle to become more comfortable as water is the most preferred element for human agents. In subsequent steps, we can see both agents in the middle and to the top left becoming comfortable (less than 10% discomfort). One agent moves through the river until it finds a drizzle terrain on the other side of the wrapped boundary and becomes comfortable by moving to it.

For all the other agents, we do not show all simulation results but only show the final state of each agent in figure 10. All experiments are, however, run until completion and can be viewed by watching the recorded videos or using the *.log01* files provided in corresponding folders inside the *results* folder. In each image, the terrain view corresponds to that agent's level of comfort in each cell, and the original terrain is also provided for easier comparison. The summary of the results are as follows:

- Agent01: Drizzle terrains seem to be the most comfortable location for human agents. Eight agents out of ten survive and 100% of the surviving agents become comfortable.
- Agent02: Interestingly, fiery monster agents prefer lava cells over heated cells due to their aversion to water present in such cells. Their high fire-tolerance seems to make both volcano and lava cells comfortable for them, with lava cells being slightly more comfortable. The monster in the river dies at the beginning and out of the remaining nine, 88.8% cells end up in a comfortable state.
- Agent03: For vapor aliens, tornado cells are deadly, causing the fatality of 3 agents at the beginning. There are no cells where these agents can feel comfortable, and hence, all seven remaining aliens stay uncomfortable by the simulation end.
- Agent04: These agents show very similar behavior to human agents and end up preferring drizzle cells over others. However, since there is no strong discomfort felt due to any element, no agents die at the beginning, and 90% of the cells become comfortable.
- Agent05: These agents end up occupying lava cells exclusively due to their low pressure and wetness value. Although all agents survive, none of them are able to reach a comfortable state.
- Agent06: The preference of these agents is all-over the place but they only become comfortable when occupying breeze cells. All ten agents survive and 40% of them end up comfortable.
- Agent07: As we can see from the high density of yellow cells in the terrain, these agents don't particularly feel

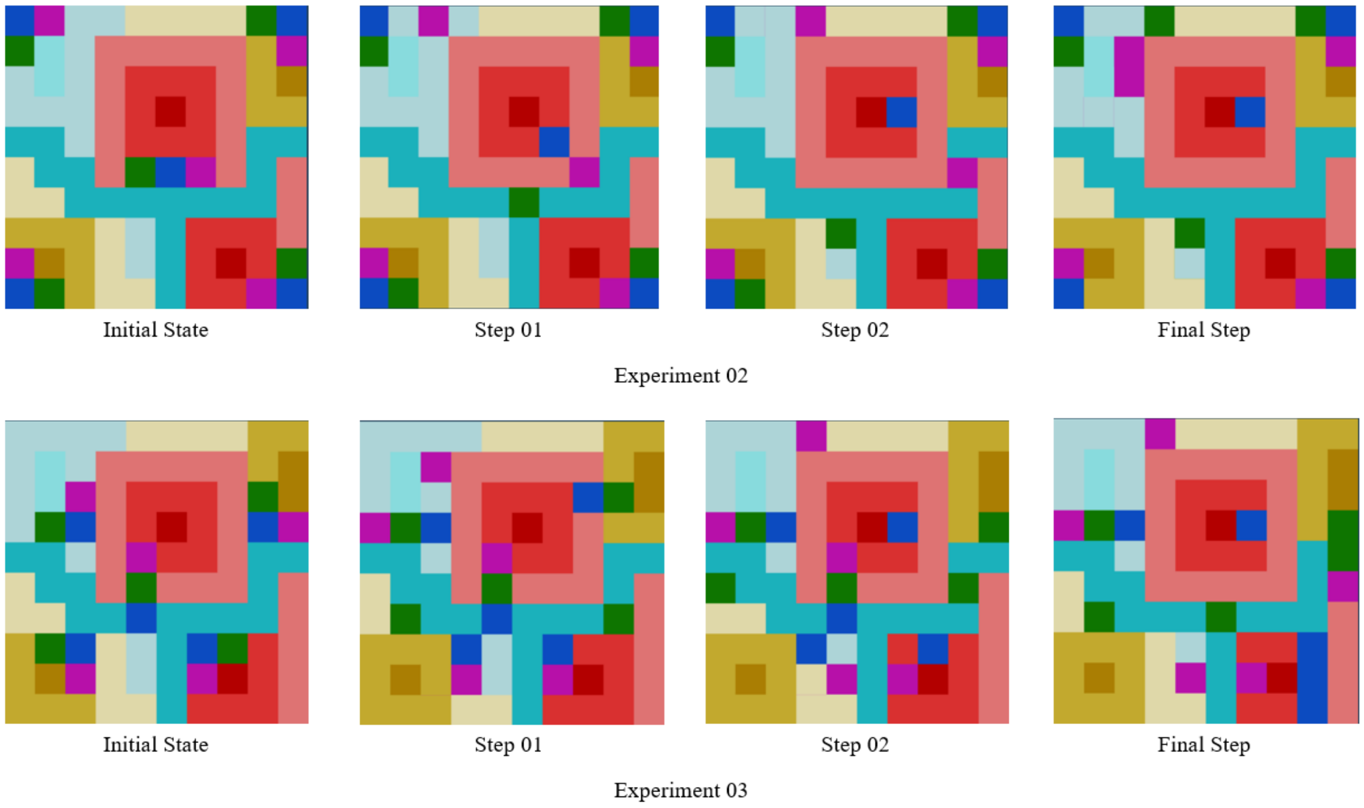


Fig. 11. Experiments 02 and 03 with multiple reactive agent types

uncomfortable in any of the cells. However, this also means that the reverse is true, and there is not a single cell where these agents are able to become comfortable. As such, all surviving ten agents remain uncomfortable.

Overall, this experiment allowed us to see some interesting behavior from different agent types and the flexibility to model different agents in varying simulation terrains.

C. *Simulating Multiple Comfort-driven Agents*

After successfully simulating the reactive behavior of one agent type, we now focus our attention on modeling multiple instances of our three primary agent types; humans, fiery monsters, and vapor aliens. The `agent` state variable is now able to hold two more values, 2 and 3, representing monster and alien agents, respectively. The previously discussed `comfort` port has been repurposed as the `comfort_human` port, and two new ports, namely `comfort_mons` and `comfort_alien` has been added to allow each cell to output its COMFORT_LEVEL for a human, monster, or alien agent to all neighboring cells.

The death rule is removed as we do not want agents to die anymore rather want to see how they interact when different types are placed together. A set of 16 identical rules have been added to model the eight-way directional movement of both monster and alien agents, taking the total number of rules in our *reactiveagent.ma* file to 49 rules. While trying to move to a location using the same directional rule, human agents now take priority over monster agents, and monster agents are prioritized before alien agents. The macros have been updated accordingly to deal with the COMFORT_LEVEL of both monster and alien agents. All files relating to this model can be found in the *comfort-driven_multi_agents* folder inside the root project directory.

We perform two experiments with several units of multiple agent types placed together. Figure 11 presents 15 agents with five units of all three agent types scattered throughout the game terrain. Human, monster, and alien agents are depicted in green, dark blue, and magenta, respectively. The game terrain retains its original terrain colors as it is not possible to show the COMFORT_LEVEL of all three agent types at the same time. For concision purposes, we only show the initial, final, and first two steps of each experiment. Detailed results can be obtained by running the same simulations or observing the recorded videos in the *results* folder.

Experiment 02: Five trios of all three agent types are placed in the four different corners as well as the middle of the map. During different steps, although the agents in the middle start moving based on their own comfort in certain cells, the agents in the corners barely shift their position, and eight out of ten corner agents retain their starting position at the end. This is due to the fact that all corners are connected due to our wrapped border, and the agents are negating the directional

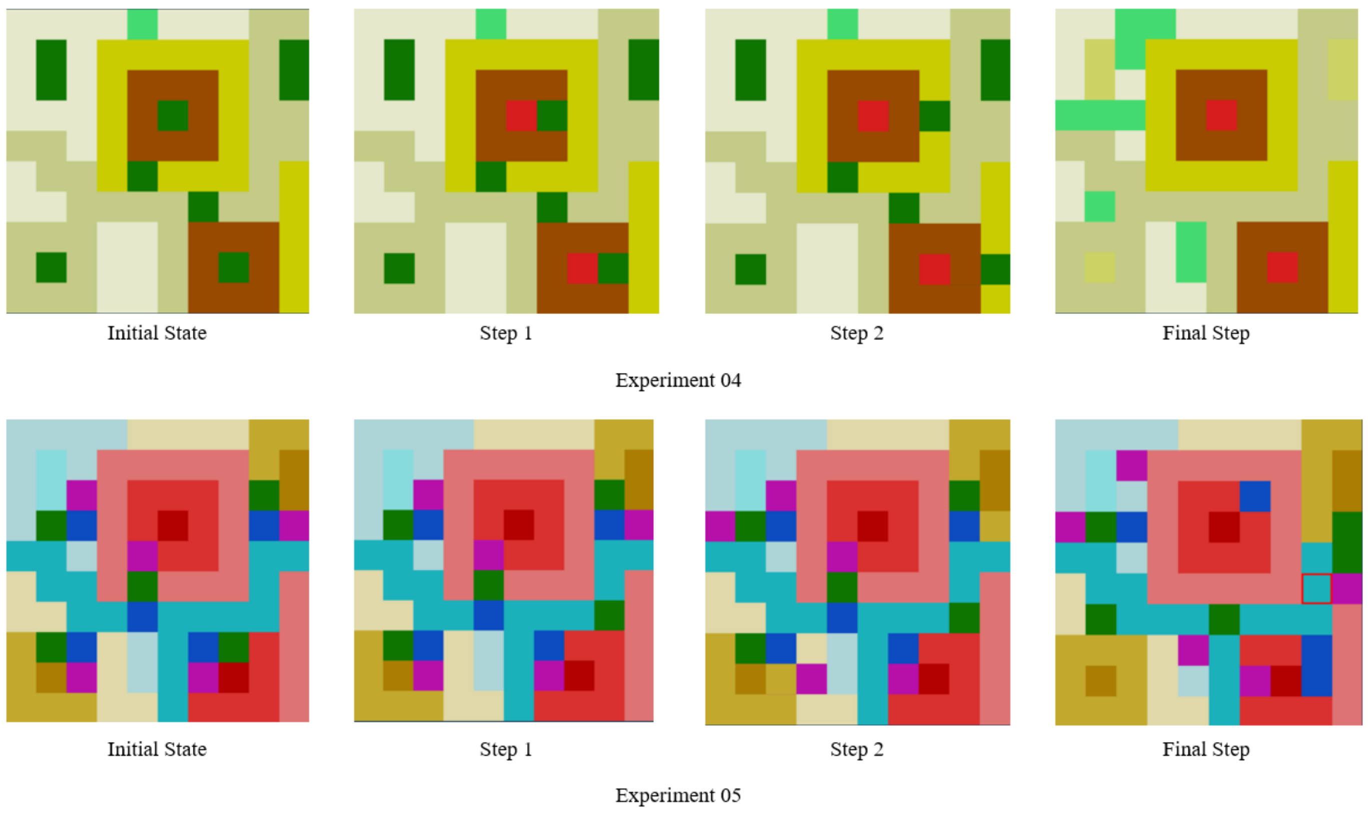


Fig. 12. Experiments 04 and 05 with varying reaction speeds

movement of each other due to being too close together.

Experiment 03: Five trios of all three agent types are placed again on the map. However, the difference this time is in the placement change of the corner agents, allowing them more room to move around. This results in a vastly improved behavior in the directional movements, and all but two agents end up moving to different positions than their starting position.

Overall, these experiments allowed us to observe the reactive movement of multiple agent types simulated simultaneously. While the model can be easily scaled to include as many agent types as we want, careful consideration must be placed in the number of rules and the priorities, as the inclusion of a single agent type involves adding, at a minimum, 16 directional rules, and redefinition of the priorities.

D. Simulating Reaction Speed

In all of our previous experiments, the state of all cells was evaluated synchronously at 100 milliseconds (ms). However, in a real scenario, a comfortable and uncomfortable agent will not move at the same speed. A great example of this behavior can be observed in the Stronghold Crusader[2] game franchise. In these games, workers under the player's control would work less and move slowly if they were to live under a king who was kind to them, fed them full meals, and prepared entertainments for the citizens. This would make the workers sluggish as they felt more comfortable in their surrounding environment. However, under a king who would torture them and fed them only in exchange for strenuous work, the workers would always complain but work or move much faster than normal. Additionally, normal citizens would idly move around a city but run screaming with fear if their houses were to catch fire or under enemy attack. All of these strongly suggests the high impact of discomfort on reaction speed.

[2]https://fireflyworlds.com/games/strongholdcrusader/

TABLE III
AGENT REACTION SPEED BASED ON COMFORT

Discomfort	Agent State	Reaction	Reaction Time (time units)
below 10%	Comfortable	Slowest	100
10% - 50%	Slightly Uncomfortable	Slow	70
50% - 80%	Highly Uncomfortable	Fast	30
above 80%	Fatally Uncomfortable	Immediate	10

The authors of [1] discuss the concept of reaction speed based on current cell discomfort. Table III shows the different reaction times of agents to be modeled when feeling different levels of discomfort. Our agents with improved reaction times now react the slowest (100 ms) and move only when they need to when they feel comfortable (less than 10% discomfort) in their current cell. Agents react faster when they start feeling uncomfortable and try to move almost immediately (10 ms)

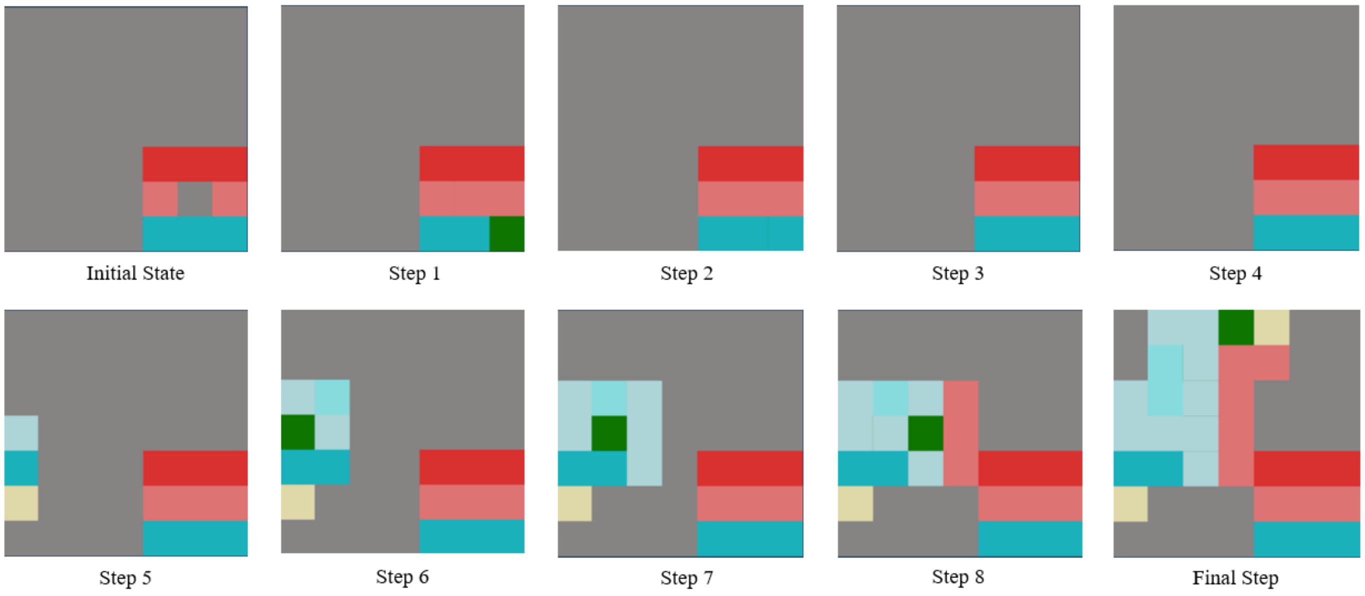


Fig. 13. Experiment 06: Single Reactive Explorer (AREA)

when above the extreme discomfort threshold (80%). All agent rules from section IV-E have been updated to include this timing behavior of agents. All files related to this model can be found in the *comfort-driven_reaction_speed* folder in the root project directory.

Figure 12 shows two simulation scenarios for experiments 04 and 05. The details of the experiments are as follows:

Experiment 04: This experiment is done with ten human agents with initial agent positions similar to experiment 01, and the game terrain is depicted from a human agent's POV. While the Insta-death rule is brought back, we can see that before any agent can move, the two most uncomfortable agents in volcano cells move away in the first two steps and survive. As expected from our rules, step 1 is evaluated at 10 ms, and step 2 is evaluated at 40 ms (30 ms after step 1) from the start of the simulation due to the two agents feeling either fatally or highly uncomfortable.

The state of the cell space is no longer updated at the usual 100 ms synchronously but is updated at 10, 30, 70, or 100 ms depending on the discomfort felt by individual agents. This does, however, increase the simulation time as many more time steps are needed to take into account and sometimes leads to wasted evaluation when no agent movement occurs. Additionally, somewhere along the ninth step, one agent at the top-left mysteriously disappears, and there is an empty white cell (comfortable for human agents) left unoccupied by all surrounding agents, which we could not explain. We imagine that the disappearance is due to the movement rules being evaluated asynchronously, and two cells based on their current COMFORT_LEVEL is passing their agents, while only one can be accepted by the receiving cell. Sometimes running the simulations created only one log file without any results, however, if we changed the simulation command to change the log file names from the default "reactiveagent.log" to anything else, the log files were created properly.

Experiment 05: This experiment is done with an even more granular time step and with all 15 agents placed as in experiment 03. The updated rules dictate that an agent's current discomfort level determines how fast they should move based on a default 100 ms time step. An agent that is feeling perfectly comfortable (0% discomfort), moves at 100 ms. Higher discomfort levels result in faster movement. For example, an agent with extreme discomfort at 93% moves at seven time units (100 - 93). As such, there can be a total of 101 different reaction speeds (only integers) all agents can have based on their COMFORT_LEVEL in the current cell. Files related to this model can be found in the *comfort-driven_speed_granular* folder inside the root project directory.

With this highly granular approach, almost every evaluation resulted in individual agent movements. In step 01, a human agent moves after 38 ms whereas, in step 02, two alien agents move after only two more ms later. After some initial timely movements, the state of the system is updated at every one ms with a lot of wasted evaluations. Running the simulation for only two seconds generated a significantly large log file that is approximately 11 MB in size (only provided in the *Exp05* folder due to its large size). At one second and 110 ms, a monster agent at the bottom-right disappears without any apparent reason again.

Overall, Experiments 04 and 05 allowed us to visualize reactive behavior with varying reactive speeds. However, this also led to many unneeded evaluations, and in some cases, disappearing agents. Due to the large file size and simulation times, coupled with the inexplicable situations, we use the default time delay of 100 ms in all future simulations.

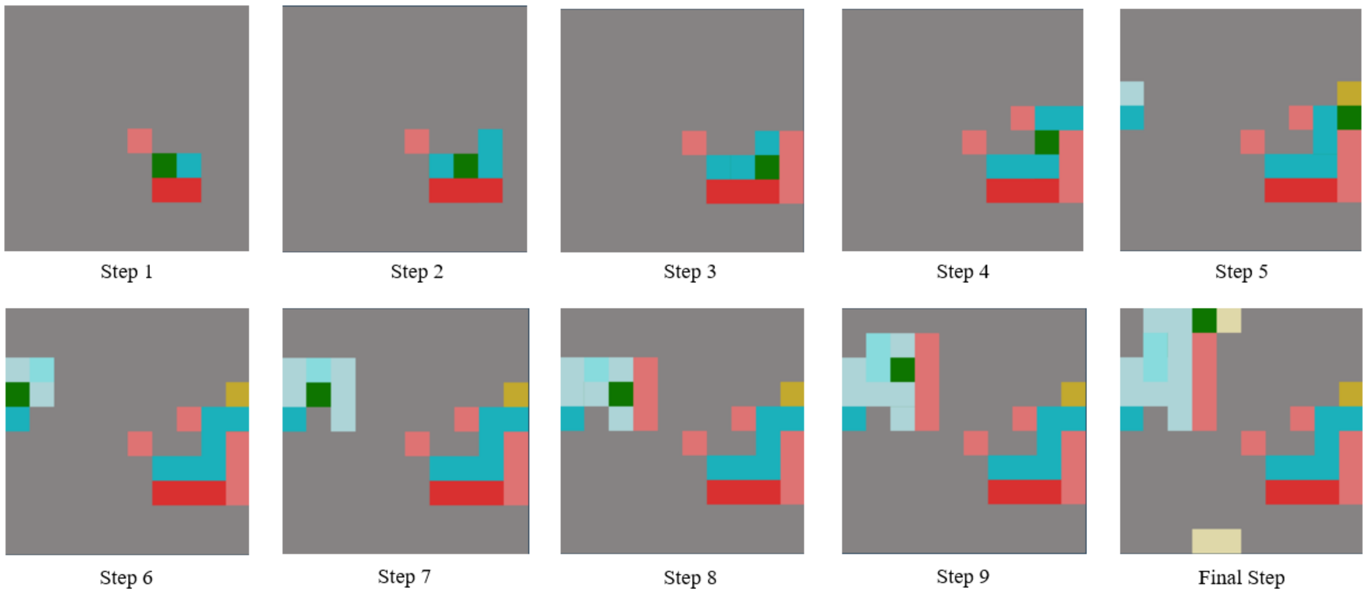


Fig. 14. Experiment 07: Single Reactive Explorer (CONE)

VI. Simulating Reactive Explorers

In many games, the game terrain is not immediately visible to the agents unless they have access to a map. Many strategy games, namely the Age of Empire and the Age of Mythology franchise, rather start with unexplored maps at the beginning. Only those portions of the terrain where there are buildings or agents under the player's control are visible to the player. Unexplored areas can be explored by moving agents under their control through those areas, and the revealed area depends on the visual capability of the explorer agent. In this section, we aim to model such exploratory behavior of agents with two different visual capabilities; area and front-cone exploration.

A. The Reactive Explorer Model - AREA Exploration

To add the area exploration capability to our agents, we replace our Insta-death rule with the Area Exploration rule. This rule states that every time an agent occupies or moves to a cell, reveal all unexplored cells in its immediate neighbor. This elevated visual capability is similar to an agent that is moving in an airplane or using a glider where they have a bird's-eye view of the landscape. To deal with the unexplored state of a cell, we add a new state variable `visibility` that keeps track of explored and unexplored cells. All cells in the map start as unexplored (`visibility` set to 0) and become visible when explored by a neighboring agent (`visibility` set to 1). A new neighbor port `explored` outputs the visibility status of a cell to its neighbors. All files related to this can be found in the *reactive_explorer_area* folder.

Experiment 06: Figure 13 shows the reactive exploratory behavior of one single human agent. Gray cells represent unexplored areas and turn into cells matching those of our original game terrain as they are explored. The agent is placed at the cell with coordinates (5,5) at the beginning, and although the cell cannot be seen due to visualization issues, the explored neighborhood around the cell can be seen clearly. While visualizing this we ran across a problem where the cell space was truncated to a 7x7 cell space without any apparent reason. To investigate this, we manually checked the logfile generated but found no issues with it. The model still performs exactly as expected, however, some simulations cannot be visualized as represented by steps 2, 3, and 4. Random changes in the model would sometimes increase or decrease the cell space shown, and we include these images here to highlight this problem with the visualization tool that might need further investigation.

By step 5, the agent appears on the other side of the model, and we can see our Area Exploration rule in effect again. For each subsequent step, the agent moves based on his comfort, every time revealing all surrounding areas around them. At the 11th (final) step, the agent explores all the areas that it could explore comfortably, and moves back and forth between the last two comfortable positions.

B. The Reactive Explorer Model - CONE Exploration

While the bird's eye view was very interesting, real human agents would hardly have access to such an elevated moving capability. And even if they did, the idea of feeling comfortable becomes a moot question at that point. If we think and about a real human surveyor, they would only be able to survey areas of the map in front of them within the human field of view (FOV). Typical human FOV is around 135°, and to model the same, we give our agents the capability to explore cells that is directly or diagonally in front of them. However, to add this front-cone exploration capability, we needed to know the directions agents were facing.

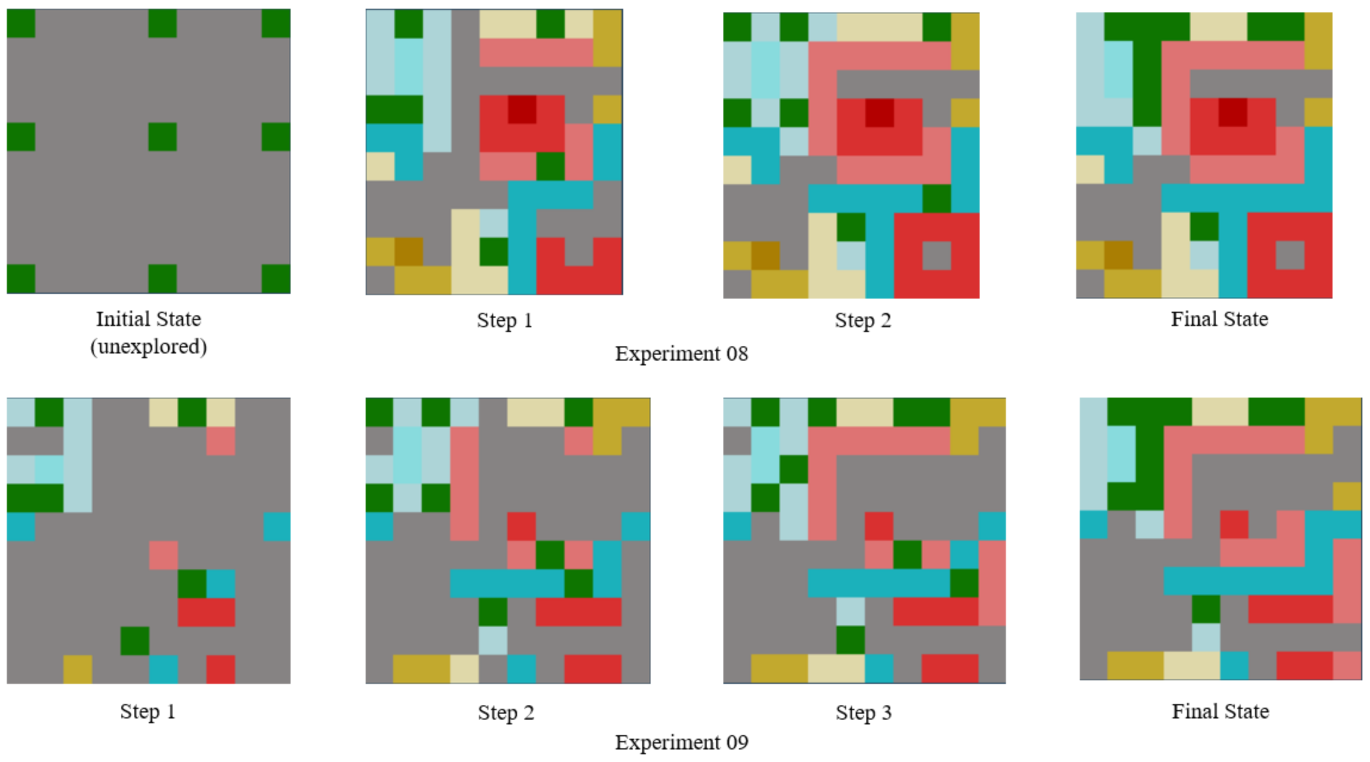


Fig. 15. Multi-Agent Exploration

We added a new state variable `direction` that holds values representing an agent's faced direction. The values 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008 correspond to an agent facing East, Northeast, Southeast, North, South, Northwest, Southwest, and West, respectively. A new neighbor port, `facing`, was used to broadcast the direction of an agent to all neighboring agents. All movement rules were updated to include these new additions, so any agent movements are also associated with the correct directional assignment. The Area Exploration rule was replaced by the Cone Exploration rule that allows agents to explore the cells that are directly or diagonally ahead of them based on the direction that they are facing. All files related to this can be found in the *reactive_explorer_cone* folder inside the root project directory.

Experiment 07: Figure 14 shows the CONE exploration behavior of reactive agents. Fortunately, this time we can see the entire cell space, and thus the proper steps followed by the agent. The agent starts at the same location as our previous experiment (not shown), and since the direction is unknown at the beginning, it is only capable of exploring that cell. At step 1, the agent moves diagonally down from the heated cells to the river to feel more comfortable. This Southeast movement made the agent face southeast and revealed the Southeast (direct facing) and the East and South neighbors (diagonal facing) to that agent. At step 2, the agent moved directly East, faced East, and revealed the two unexplored cells directly and diagonally ahead. This process continues until the agent reaches the same cell as experiment 06 in step 11, and we can see the CONE exploration behavior throughout the rest of the images.

C. AREA and CONE Exploration - Multiple Agents

After testing our AREA and CONE Exploration rules for single agents, we move towards simulating multiple agents. Figure 15 shows the initial state of the nine human agents placed in the game terrain; four agents occupy the four corners, four others occupy the cell in the middle of the borders, and one additional agent occupies a cell in the middle. We perform two experiments with this setup and try to see the flocking behavior of agents based on their comfort. In each case, we calculate the percentage of cell space explored by both AREA and CONE Explorer agents.

Experiment 08: In this experiment, the behavior of nine AREA Explorer human agents are shown in different steps. We run into the same problem as experiment 06, except that fact that the cell space, during visualization, is truncated to 10x9. Nevertheless, out of the visible cells, 81% of the cells become visible after all agents finish their exploration and end up flocking to the top-left drizzle cells of the map.

Experiment 09: This experiment tries to see the behavior of nine CONE Explorer human agents. While the agents end up in the same positions, the area explored compared to the AREA Explorer agents is significantly less due to the agents not having sideways or backward visual capability. In the end, 61% of the terrain becomes visible to the agents.

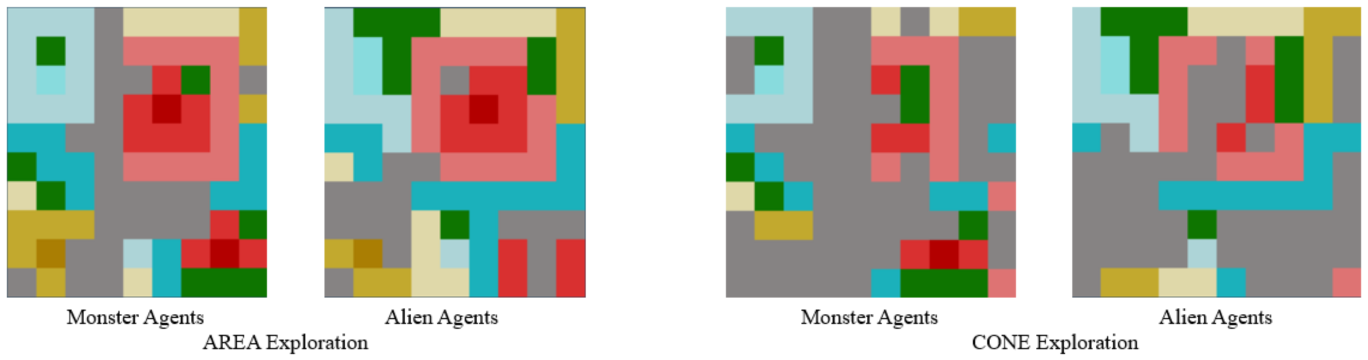


Fig. 16. Experiment 10: Exploration by other primary agents

Experiment 10: Figure 16 shows the final areas explored by other primary agents with the same initial nine-agent setup. Out of our three primary agents, Monster agents explore the least area (75%) with AREA exploration and moderate area (48%) with CONE Exploration capabilities. This is due to the fact that they found easier access to fire-attribute cells from their initial positions and didn't have to move much to stay comfortable. Alien agents, on the other hand, end up exploring 82% (highest) of the map with AREA Exploration and 43% (lowest) with CONE exploration capabilities. The lowest value with CONE exploration can be attributed to agents moving towards the same direction with less changes in agent-facing. Since the direction that the agent is facing does not affect the bird's-view perspective, this has no effect on AREA exploration at all.

Overall, experiments 06 to 10 allowed us to see the interesting exploratory behavior of both single and multiple agents placed in the game terrain and observe the relation between agent facing changes during movement and the area explored due to AREA and CONE exploration capabilities.

VII. Simulating Desire-Driven Agents

The reactive behavior of agents observed so far has been purely based on how comfortable they feel when standing atop a cell. However, we must not forget that depending on the agent's job or mission, they might display entirely different behavior. A woodcutter, for example, might need to go to a jungle during the daytime to cut woods, even though they are likely to be attacked by wild animals on the way. The desire to do his work has a clear contention with how comfortable he feels doing his job or while moving to his goal. Soldiers on a mission might entirely disregard their feelings of discomfort and focus on their goal of capturing the enemy base only. During this mission, they might have to go over dangerous terrains or the battlefield where they might instantly die, but they do it regardless due to a very high desire to achieve their objective. In this section, we investigate the interplay between desire and comfort and how agents with different levels of those parameters react.

A. *The Desire-Driven Agent Model*

We update our model to add the capability of agents to make decisions based on both desire and comfort. An additional state variable called `inclination` now takes into account both the COMFORT_LEVEL and Desirability of agents, and also the weights assigned to these parameters to determine the course of action for an agent. COMFORT_LEVEL is calculated as before, whereas Desirability is calculated from the heretofore unused `desirability` variable. After assigning suitable cells in the map certain `desirability` values, `inclination` is calculated as follows:

$$I = CL * W\_C + D * W\_D$$

W_C and W_D represent the weight of comfort and desire, respectively, for the specific type of agent being modeled. Based on this idea of weights, table IV presents the five different types of agents that we experiment on. All of the experiments will be performed using a human agent with COMFORT_LEVEL as defined in table II. All experiments done until now have been done with purely reactive agents (R-01) that do not respond to the desirability of cells, and as such, we do not show results for these agents any further. We create a new macro file *macro_weights.inc* where these weights can be easily manipulated to model the same and more types of agents as table IV.

TABLE IV
Preference Percentage between Comfort and Desire

Agent Type	Agent ID	Comfort Preference	Desirability Preference
Purely Reactive	R-01	100%	0%
Highly Reactive	R-02	75%	25%
Evenly Weighted	R-03	50%	50%
Highly Goal-directed	R-04	25%	75%
Purely Goal-directed	R-05	0%	100%

Highly reactive agents (R-02) prefer comfort over desire, whereas highly goal-directed agents (R-04) prefer the opposite. Evenly weighted agents (R-03) take both equally into account, and purely goal-directed agents (R-05) react based on purely their desire. We remove all variables and neighbor ports

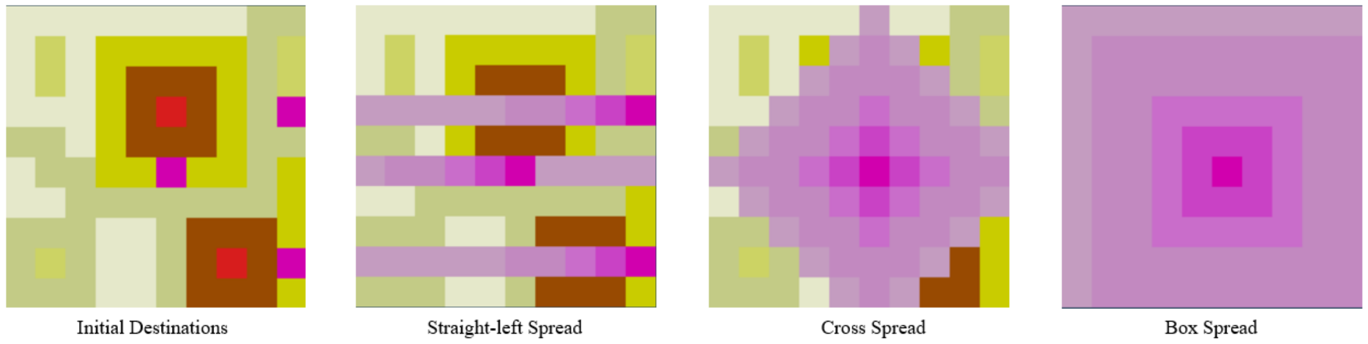


Fig. 17. The spread of desire

needed for exploratory behavior and add a new port named `preference` to output the inclination level of a cell to their neighbors. All movement rules have been updated to include the use of the `preference` port, and agents now react and make decisions based on their preference for a certain cell (lower is better). The exploration rules have been replaced with the Insta-death rules again as we wish to see whether, and at what level of desire, agents are willing to give their lives to reach their goal.

B. Spreading Desire

The spread of desire from the original objective is done in three ways, as illustrated in figure 17. We define three new macros in the file *macro_desirability.inc*. The first macro SPREAD_DESIRE_STRAIGHT takes the desirability values directly from the cell to their right as long as they don't have any assigned desirability. The desirability gained is based on a propagation constant, which by default, is set to 0.7 as it allows maximum desirability to travel from one end of the map to the other without reducing to absurdly low numbers. The first two images in the figure represent this spread behavior where dark pink is used to represent the objective, and reduced levels of pink are used to represent a reduced level of desire. We only consider a straight-left spread as our agents have a right moving behavior, but this can be done in any single horizontal, vertical, or diagonal direction by modifying the associated macro.

The second macro SPREAD_DESIRE_CROSS spreads desire using the Von Neumann neighborhood, whereas the third macro SPREAD_DESIRE_BOX does it using Moore's neighborhood. Examples of these two spreads are presented in the third and fourth image of figure 17 with only a single centered objective. During our experiments, we only use one of these three macros defined as another macro SPREAD_DESIRE. The type of spread, as well as the propagation constant, are easily modifiable and gives the modeler great flexibility in choosing the scenarios that they wish to model. All files related to our desire-driven model

C. Simulating Straight Spread of Desire

With a set of experiments, we now try to see the behavior of our desire-driven agents in different comfort versus desire scenarios. All experiments consider the same initial starting position of ten human agents as experiment 01, as we wish to see agents in extremely uncomfortable situations and the effectiveness of the Insta-death rule at the start and with the passage of time. Figure 18 provides an elaborate depiction of simulations done with agent types R-02, R-03, R-04, and R-05.

Experiment 11: All of these simulations correspond to experiment 11, where we try to see the behavior change in human agents as their preference changes due to the increasing desirability of cells over time. The game terrain shows the preference from a human agent's POV and uses the same color profiles as was used in experiment 01 to depict COMFORT_LEVEL. This experiment has three objectives with maximum desirability with a position identical to the initial destinations shown in figure 17. Over time, desire from objective cells spreads through the cells to the left, which can be seen by those cells taking lower (more desirable) preference values. The two objectives at the right border require agents to walk over volcano cells in order to reach them. We wish to see whether agents are willing to do that, and if they are, what distribution of desirability versus comfort forces them to do so.

- Type R-01 Agents: These agents react purely based on their comfort (100%), and as such, no spread of desirability have any effect on them. The results from these agents are exactly the same as figure 8 (experiment 01), and therefore they are not simulated, nor shown, for experiment 11.
- Type R-02 Agents: These agents are highly reactive (75% comfort, 25% desire), and produces results similar to Type R-01 agents such as two agents standing on volcano cells dying at the beginning and most agents flocking to the top-left drizzle cells. The big difference is, however, seen in agents who are close to the highly desirable objectives. One agent at the middle does not move into the river but prefers to stay on the objective (heated cell) due to the high preference resulting from high desire values. Another agent that started in the river, does not move through the river rather waits for the destination to be unoccupied to move to that. This, however, never

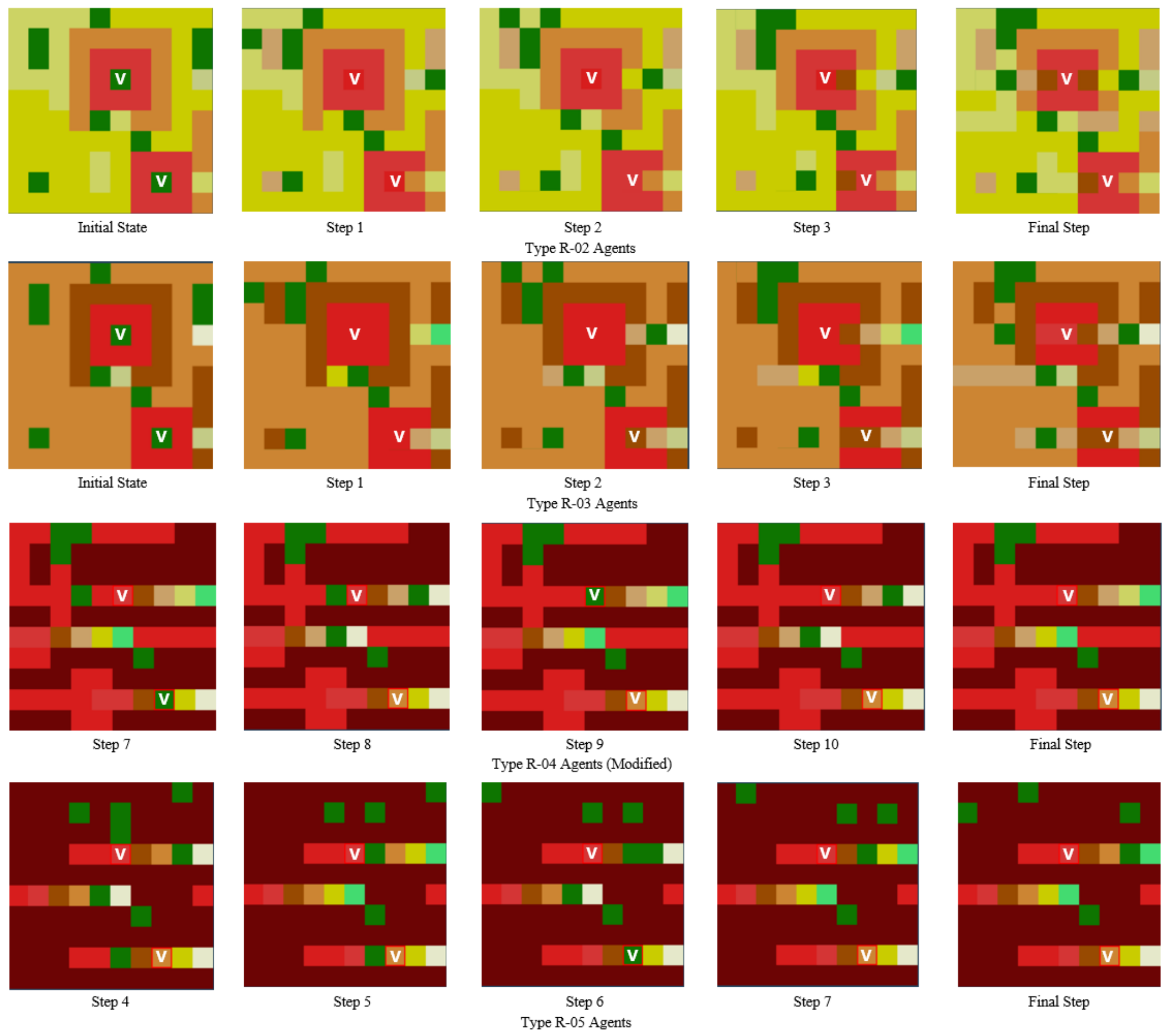


Fig. 18. Desire-driven human agent behavior (Straight-left spread)

happens due to Movement rule 1 taking priority over Movement rule 6, and that agent waits forever on that cell. One other agent at the top-right stays close to the highly desirable target, whereas another agent at the bottom travels about halfway from its initial position to the bottom-right objective. That agent stopping halfway is due to the desire of the next right cell not being high enough for him to move from the drizzle cells to river cells.

- Type R-03 Agents: These agents value comfort and desire evenly (50% for both) and end up in a final state very similar to that of Type R-02 Agents. One slight difference is the agent at the bottom being able to move one step to the right due to the increased percentage of desire, making that river cell the preferred destination.
- Type R-04 Agents: The original highly goal-driven agents (25% comfort, 75% desire) produce exactly the same final state as the previous experiment. Therefore, we slightly modify these agents to have a higher desire (10% comfort, 90% desire), which shows vastly altered behavior in agents. The agent at the bottom, due to its very high desire to reach the bottom target, steps over the volcano cell at step 7, while another agent at the top-left, until now stuck right before the fire-attribute cells for all experiments, takes its first step towards the top objective. The bottom agent dies at step 8, whereas the to-left agent follows the same tragic fate by stepping into the volcano cell at step 9. The final state has only six remaining agents.
- Type R-05 Agents: These agents are driven purely by desire (100%), and as such, their movement doesn't take the terrain type into consideration at all. This can be seen by the top three agents, previously stuck at drizzle cells, now moving directly right ignoring everything in their path. The bottom agent dies one step earlier at step 06. One interesting thing happens in the case of the previously dead top-left agent. When it tries to move to

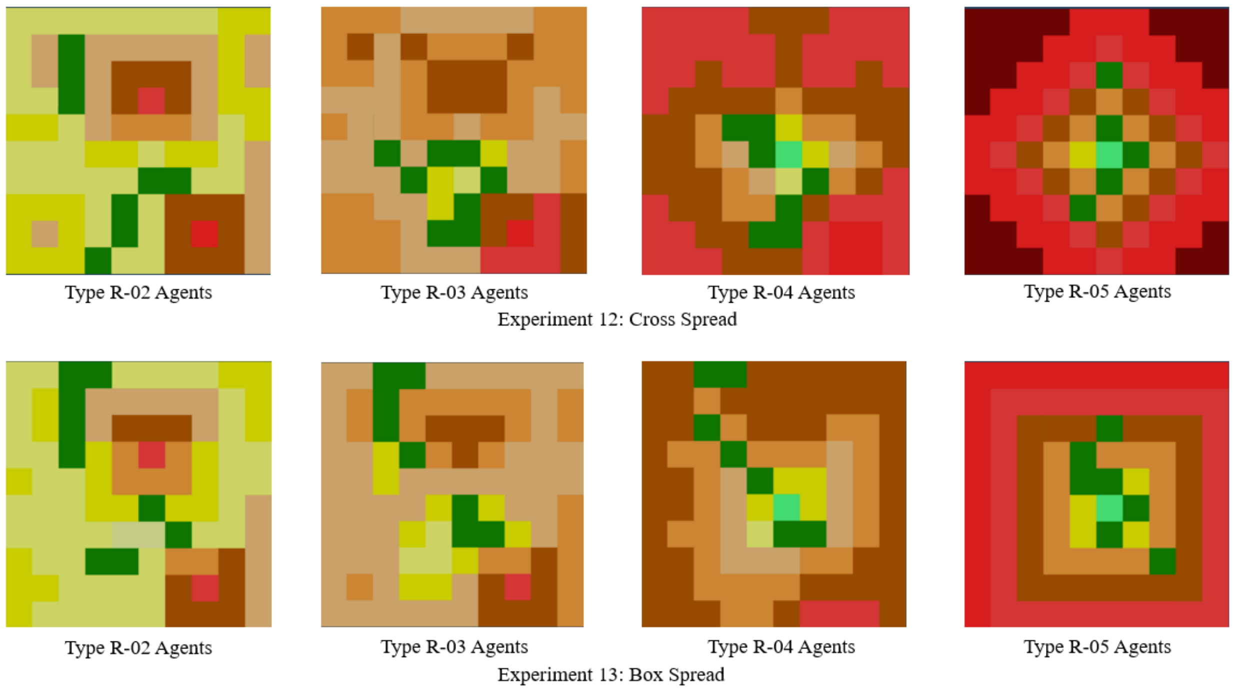


Fig. 19. Desire-driven human agent behavior (Box and Cross spread)

the path with higher desirability (step 4 to step 5), it moves past the volcano cell as the lava cell has much higher desirability due to the reduced propagation value, which allows it to survive. This is an ironic fate as it had to die when it cared about comfort, and survived when it didn't care at all, and we attribute this to the saying "fortune favors the brave" and that agent's purely desire-driven nature. In the end, seven agents out of the starting ten survive.

This series of experiments allowed us to confirm that agents are indeed willing to die for their cause if the desire to reach their objective is adequately high.

D. Simulating Cross and Box spread of desire

We perform two final experiments, experiment 12 and experiment 13, to see the effect of desire on agents with a cross-shaped (Von Neumann) and box-shaped (Moore) spread of desire, respectively. For both experiments, we use only a single objective with maximum desirability at the middle of the game terrain, at the coordinate (5,5). The initial position of the ten human agents is the same as before. Figure 19 shows the results of both experiment 12 and experiment 13. Only the final states of each simulation for every agent type are shown.

Experiment 12: We simulate ten human agents with desire spreading from the objective in the middle forming a Cross shape. A brief discussion of the observations are as follows:

- Type R-02 Agents: Five out of eight surviving agents gather around the middle destination cell.
- Type R-03 Agents: All surviving eight agents seem to flock to the destination point.
- Type R-04 Agents: Similar results as Type R-03 Agents. The exact destination point is now occupied.
- Type R-05 Agents: Two agents die before they could be attracted to the destination due to dangerous right-moving behavior without considering terrain types. Four agents almost form a perfect cross at the middle.

Experiment 13: This experiment is performed with agents located as in experiment 12 but with a box-shaped spread of desire. Following are some of the more interesting results:

- Type R-02 Agents: Very similar results as experiment 01 with no desirability. Only four agents with starting positions close to the objective manage to stay nearby.
- Type R-03 Agents: Interestingly, only three agents flock to the objective, but they form a tighter group. A slight change is seen in one of the agents previously grouped at the top-left.
- Type R-04 Agents: About four agents occupy the first level of Moore neighborhood.
- Type R-05 Agents: No agents die due to desire spreading much faster throughout the map with this spread. Five agents manage to enter the first level of the Moore neighborhood, and all but one agent manage to get into the second level.

Incomplete Experiments: These are some of the results from experiments that we could not manage to test extensively due to a shortage of time. We do not provide the models or any simulation results in our project folder due to those not being thoroughly tested. A brief description of some of the preliminary results are given below:

- Experiments with an increased propagation constant: The propagation constant has been increased from 0.7 to 0.8 to have a stronger overall propagation of desire. This results in type R-02 and type R-03 agents responding much better to spreading desirability as they prefer to move to uncomfortable and/or potentially fatal cells much earlier than experiments 11-13.

- Experiments with preset desirability in the *reactiveagent.val* file: This corresponds to agents having a map of their desired destination at the beginning. This does not result in particularly interesting behavior but does prevent the two agents from dying in the case of Type R-05 agents in experiment 12, as desire is fixed and does not need to propagate with time.
- Destinations with increasing desirability over time: This, in turn, increases the values of the neighborhood desirability as well, and if simulations are run for a long time, can result in negative `preference` values. We had a lot of problems with visualizing this as desire grew indefinitely and overwrote agents during visualizations. We did not manage to fix dealing with this problem due to a shortage of time.

Overall, experiments 11 to 13 allowed us to see the interplay between agents reacting to different levels of comfort and desire. We found that with high enough desirability, agents are indeed willing to give their lives if they could reach their destination. Even with very high desirability in the middle, not all cells were occupied. We surmise this is due to the Movement rules preventing movement when the most preferred cell is occupied by another agent, as can be seen clearly in the diagonal line formed by type R-04 agents in experiment 13.

VIII. Challenges Faced

We faced numerous challenges while doing the project. These challenges resulted primarily from rule creation, error handling, and visualization issues.

While the authors of [1] discuss the general ideas of COMFORT_LEVEL and Reaction Speed, they provide no concrete rules. All movement, death, exploration, and comfort versus desire management rules had to be written from scratch, requiring a lot of imagination, as well as many trials and errors. Error handling was a key issue during writing rules and led to many hours spent debugging syntax errors. A clear example to show this problem would be the fact that `agent!=1` would work, but `agent=1` or `agent= 1` would not work without space before the equal sign within if statements, which was very easy to go unnoticed. Since the software only provided a generic error message saying there was an error in parsing the rules, we had to manually check most parts of the code, which in many cases required checking hundreds of lines. We believe that at least an indication of the line number in which the error had occurred may be a great addition to the already fascinating CD++ tool.

We also faced a variety of challenges during the visualization. We could not manage to make Drawlog, the DEVS Web Viewer link[3] provided in the "WebViewerTutorial.pdf" or the link[4] hosted in GitHub work with the results of our simulations. Before we found the ARSLab Simulation Viewer, we were manually checking the log files to ensure that the model was working as intended. While this could very well be due to our own limitations, it resulted in hours of frustration.

Changes in the model would sometimes turn the cell space into a truncated version of the original with less height or width or both during visualization. This problem made observing and verifying results much harder to do. Something in the model had to change for this to be fixed, which was not readily apparent. In many other cases, a simple change in the model would make a working .json file suddenly not work anymore. We had to either find and revert the changes or make a new version of the same file with the same parameters, leading to redundant work. In the case of manually creating .val files, if a newline was not put after the ending cell value (e.g., (9,9) in case of a 10x10 cell space), that value would not be registered.

In the original work of Sweetser and Wiles [1], the authors used the Von Neumann neighborhood to model four directional movements. We decided to use the Moore neighborhood for improved eight-directional movement capability. This choice, however, made adding any additional rule and debugging much harder. As part of the project, we wanted to see what would happen if agents could determine their next destination based on the second level of the Moore neighborhood. However, this would entail using the third level to stop agents from colliding, making this highly complex task infeasible within the time that we had left at the end. Additionally, visualizations were not straightforward as agents were not simply occupying empty cells or avoiding obstacles but could occupy cells with multiple state variables and values such as `comfort` or `preference`. Before we figured out exactly what was being shown in the single visible layer, this resulted in hours of redoing things to create proper visualizations.

All in all, this project required an unimaginable amount of effort and countless trials and errors. However, throughout this taxing process, we gained a tremendous amount of knowledge and experience, and the final project turned out to be something we can take pride in.

IX. Conclusion

In this project, we provided a cellular automata model of reactive game agents in the Cell-DEVS modeling formalism, where agents are capable of reacting to changes in their environment based on their comfort and desire. To verify the correctness of the model, we performed a set of 13 experiments demonstrating the comfort-driven, exploratory, and desire-driven behavior of both multiple agents in a fixed game terrain containing nine different terrain types. Additionally, we investigated variations in reaction speed and analyzed the

[3]http://ec2-3-235-245-192.compute-1.amazonaws.com:8080/devs-viewer/app-simple/

[4]https://staubibr.github.io/arslab-prd/app-simple/index.html

dynamic behavior of agents that resulted from the interplay between comfort and desire.

Our experiments proved that agents are able to react to their surrounding environment by determining the best course of action based on where they felt more comfortable or which direction led them to more desirable goals. The timed reaction of agents based on their comfort in the current cell led to cases where agents mysteriously disappeared, a phenomenon we were unable to unravel.

The exploratory behavior of agents led to intriguing results where the same agent with the same cells traversed produced the highest explored area in case of area exploration and the lowest explored area in case of frontal-cone exploration due to their facing during the movement. Moreover, we determined that agents with a highly desire-driven nature are willing to ignore their discomfort and walk to their death if that meant that they would reach their desired objective.

The experiments performed in this project allowed us to dive deep into the dynamic behavior of different game agents and gain insight into modeling highly intelligent reactive agents. The study presented could lead others to investigate various other phenomena with game agents with additional capabilities and discover hidden, unintended, or undesired interactions among reactive game agents.

Our future work will focus on modeling reactive agents with increased neighborhood visibility and explore the interplay between comfort and desirability when the agents are aware of their destination from the outset.

References

[1] P. Sweetser and J. Wiles, "Combining influence maps and cellular automata for reactive game agents," in *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 524–531, Springer, 2005.

[2] D. Carmel and S. Markovitch, "Learning models of intelligent agents," in *AAAI/IAAI, Vol. 1*, pp. 62–67, 1996.

[3] K. T. Gon, B. P. Zeigler, and H. Praehofer, "Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems," 2000.

[4] G. A. Wainer and N. Giambiasi, "Application of the cell-devs paradigm for cell spaces modelling and simulation," *Simulation*, vol. 76, no. 1, pp. 22–39, 2001.

[5] G. Wainer, Q. Liu, O. Dalle, and B. P. Zeigler, "Applying cellular automata and devs methodologies to digital games: A survey," *Simulation & Gaming*, vol. 41, no. 6, pp. 796–823, 2010.

[6] G. Wainer, "Cd++: a toolkit to define discrete-event models," *Software, Practice and Experience*, vol. 32, no. 3, pp. 1261–1306, 2002.

[7] A. López and G. Wainer, "Improved cell-devs model definition in cd++," in *International Conference on Cellular Automata*, pp. 803–812, Springer, 2004.

[8] G. A. Wainer, *Discrete-event modeling and simulation: a practitioner's approach.* CRC press, 2017.