Amorphous Fortress: Exploring Emergent Behavior and Complexity in Multi-Agent 0-Player Games

M Charity
New York University
Game Innovation Lab
Brooklyn, NY
mlc761@nyu.edu

Sam Earle
New York University
Game Innovation Lab
Brooklyn, NY
se2161@nyu.edu

Dipika Rajesh
Independent
Chennai, India
dipika.rajesh@gmail.com

Mayu Wilson
Independent
Brooklyn, NY
mayuwilson@gmail.com

Julian Togelius
New York University
Game Innovation Lab
Brooklyn, NY
julian@togelius.com

***Abstract*—We introduce the Amorphous Fortress—an abstract, open-ended artificial life simulation framework. In this system, entities are represented as finite-state machines (FSMs) which allow for multi-agent interaction within a constrained space. These agents are created by randomly generating and evolving the FSMs; sampling from pre-defined states and transitions. This environment was designed to explore the emergent AI behaviors found implicitly in simulation games such as Dwarf Fortress or The Sims. We apply two evolutionary algorithms to this environment, hill-climber and MAP-Elites, to explore the various levels of depth and interaction from the generated FSMs and to generate diverse sets of environments that exhibit dynamics estimated to be complex by analyses of agents' FSM architecture and activation. This paper combines the work of two previous non-archival workshop papers.**

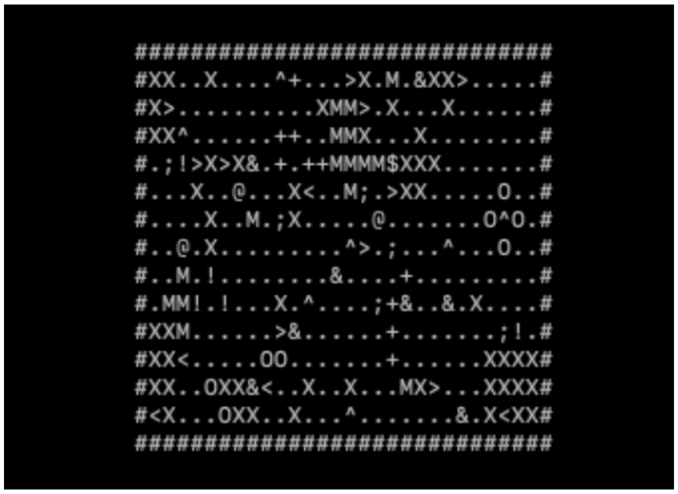
***Index Terms*—open-ended environments, artificial life, simulation, evolutionary algorithms, quality diversity**

I. Introduction

Accidental cat poisonings in Dwarf Fortress, characters from the Sims choosing to "woo-hoo" the Grim Reaper, using bullet-time and a bomb to literally fly across the map in Breath of the Wild—these are just a few examples of unintentional behaviors in games. These emergent behaviors can lead to very memorable but sometimes irreplicable experiences for the player—with the potential to become something like rumors or urban legends for the game's community. Automatically searching for emergent behaviors in games can be influential for future AI research, by defining open-ended problems, creating new interactions and spaces for multi-agent systems, and developing adaptable and unexpected NPC interactions in games.

Previous works in both research and game design have explored both simulations and emergent AIs in their own unique systems. Some examples include the computer simulation games Tierra[1], Avida[2], and Conway's Game of Life [1], each with diverse and stimulating generated narratives that stem from these AI interactions. These interactions were never originally intended or perceived of by their human designers, but nevertheless, create evocative situations and experiences within the system. Exploring these artificial experiences could allow AI designers to find interesting agent behaviors naturally without having to hard-code these behaviors within the initial system.

We introduce the Amorphous Fortress, an open-ended simulation framework for evolving artificial life with abstract and emergent behaviors. This system will allow us to observe the interactions that emerge in multi-agent simulations and how innovative and interesting behaviors can be generated from a small pre-defined set of primitive actions and conditions in a confined space. This paper combines the work of two previous non-archival workshop papers[3].



```
##############################
#XX..X....^+...>X.M.&XX>.....#
#X>..........XMM>.X...X......#
#XX^......++..MMX...X........#
#.;!>X>X&.+.++MMMM$XXX.......#
#...X..@...X<..M;.>XX.....O..#
#....X..M.;X.....@.......O^O.#
#..@.X.........^>.;...^...O..#
#..M.!........&....+.........#
#.MM!.!...X.^....;+&..&.X....#
#XXM......>&......+.......;!.#
#XX<.....OO.......+......XXXX#
#XX..OXX&<..X..X...MX>...XXXX#
#<X...OXX..X...^.......&.X<XX#
##############################
```

Fig. 1: Screenshot of a generated environment in the Amorphous Fortress framework containing multiple instances of uniquely defined autonomous agents.

For the first experiment, we conduct a hill-climbing evolutionary experiment to examine and discuss some emergent scenarios and entities with these evolved FSMs. From this, a variety of entity classes emerge, with diverse policies of agent

[1]http://tomray.me/tierra/index.html
[2]https://github.com/devosoft/avida
[3]Links removed for double-blind anonymization

behavior that depend on one another for deeper exploration of their own graphs. The second experiment implements the quality diversity (QD) algorithm MAP-Elites to evolve a grid of Amorphous Fortress environments. This algorithm allowed us to find a collection of fortresses that included entities exhibiting a variety of emergent behaviors — including those reminiscent of real-world ecosystems. The results from these two experiments demonstrate the potential and utility of the Amorphous Fortress framework as an abstract, open-ended environment for automatically generating and identifying emergent AI behaviors.

II. Background

A. Finite-State Machines

Finite state machines (FSMs)—as defined by Georgios and Togelius [?]—are an abstract representation of an interconnected set of actions, states, and transitions represented by a graph. These graphs typically represent conditional relationships between nodes and actions. FSMs are one of the key components of classic game AI. They have been used to create simple non-playable character behaviors in a large number of games, including Pac-man, Half-life, and F.E.A.R [2]. Previous works have also explored the use of FSMs as solver agents in AI competitions such as Starcraft [3] and Pommerman [4] and as agents in a mixed-initiative simulation system [5]. Despite their popularity, vanilla finite state machines lack dynamicity compared to other more recent and sophisticated approaches to NPC AI such as behavior trees. Finite state machines lack dynamicity compared to other graph-based artificially intelligent agents, such as behavior trees. They also lack in depth and complexity compared to behavior trees; which are more commonly used to create more realistic and varied NPC behaviors in games. However, the simplicity of the FSMs is precisely why we chose to use finite state machines for this system; in order to explore variance with the behaviors that could emerge from such a limited design. Here, we limit ourselves in scope to straight-ahead FSMs, with single-action nodes, and posit that applying evolutionary search to such a representation—despite its simplicity—is enough to lead to an interesting diversity of NPC behavior profiles, leaving the evolution of more general-purpose NPC AI description languages to future work.

B. Simulation games

Simulating real-world events through games allows researchers to study and emulate phenomena in controlled environments while allowing for a variety of situations and interactions internal to the systems at play. Combined with artificial agents, these simulation games can explore how AI could interact with both the player and the world itself. The SimSim environment [6] evolves furniture arrangements in houses based on the Sims games to find novel designs while keeping the player alive and satisfied. Similarly, Earle et. al [7] introduce a training environment for reinforcement learning agents in the game SimCity and examine population behaviors of cellular automata in Conway's Game of Life. Green et. al [8] use a minimal clone of RollerCoaster Tycoon to generate a diversity of theme park layouts. Thus, simulation games provide a perfect testing ground for artificial agents to either engage with or generate artifacts that can be extrapolated to real-world scenarios.

C. Emergent AI

In the game AI research field, emergent AI has become an increasingly notable area of study. The breadth of interactivity afforded by non-playable characters and back-end artificially intelligent and adaptive systems allow for more individualized player experiences, some of which the game designers may have never intended. In games, many emergent phenomena result from NPCs interacting with each other based on some set of rules, such as in Dwarf Fortress, or from adapting to player behaviors and decisions, such as the AI director in Left 4 Dead. In games research, recent work has explored how artificial generators can appear "haunted" to a player when the generator creates highly abnormal samples [9] or when narratives emerge from level generations that stem from the individual player experience and interaction with the generated artifact [10]. The focus of this paper is to explore how emergent AI behavior from generated FSMs could inspire narratives or descriptions of entities that have emerged through abstract code, or how multi-agent systems could interact without any pre-made definition to create more complex situations not foreseen by the designer.

D. Evolutionary and Quality Diversity Algorithms

Evolutionary algorithms, inspired from biology, look to evolve a population of samples towards a pre-defined and measurable objective known as a fitness function. This population is resampled and mutated to create future generations of samples that are ideally more successful than their predecessors in reaching the optimal fitness value [11]. Novelty search, one such evolutionary algorithm introduced by Lehman et. al [12], looks to maximize discovery of new or novel samples during evolution, rewarding behavior that diverges from prior behavior. Novelty search has been used to generate diverse game AI components such as video game levels [13] and dungeons [14].

This methodology has served as a precursor to another derivation of evolutionary algorithms known as quality diversity algorithms. This subset of algorithms are designed to maintain both diversity and quality across generations of populations. Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), is a QD algorithm introduced by Mouret and Clune [15] that searches a multi-dimensional search space for high-performing, diverse solutions. MAP-Elites maintains a map of the best recorded solution across a particular set of feature dimensions. MAP-Elites has shown remarkable success in generating content for games such as dungeons as shown by Alvarez et. al [16]. Guerrero and Perez-Liebana [17] utilize MAP-Elites to generate a team of agents for automated gameplay. MAP-Elites has also been used to create, replicate and explore real-world adaptability by simulated agents in

Action Node	Definition
idle	*the entity remains stationary*
move	*the entity moves in a random direction (north, south, east, or west)*
die	*the entity is deleted from the fortress*
clone	*the entity creates another instance of its own class*
push (c)	*the entity will attempt to move in a random direction and will push an entity of the specified target character into the next space over (if possible)*
take (c)	*the entity removes the nearest entity of the specified target character*
chase (c)	*the entity will move towards the position of the nearest entity of the specified target character*
add (c)	*the entity creates another instance from the class of the specified target character*
transform (c)	*the entity will change to a different entity class - thus altering its FSM definition entirely*
move_wall (c)	*the entity will attempt to move in a random direction unless there is an entity of the specified class at that position - otherwise it will remain idle*

TABLE I: Entity FSM action node definitions

virtual open-ended environment as studied by Norstein et. al [18]. This approach not only alleviates the user's workload but also enhances performance in the evaluated environments. We use MAP-Elites to generate the multiple environments and multi-agent definitions in the Amorphous Fortress that exhibit different emergent AI behaviors than what was discovered in the hillclimber experiment.

III. Amorphous Fortress System

The Amorphous Fortress[4] is an artificial life simulation system made up of a hierarchy of 3 components: entities (the agent class of the system) the fortress object (the environment class of the system) and the engine (the "manager" and main loop of the simulation). A configuration file can be provided upon initialization of the system to define a particular range of interactions and allow for simulation reproducibility in the framework. The following subsections describe each component in more detail and in the context of the experiments done for this paper.

A. Entities

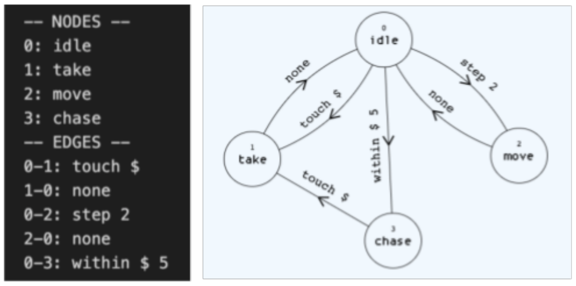
Each "entity" of the Amorphous Fortress is defined by a singular ASCII character, a unique identifier value, a list of nodes, and a set of edges. The ASCII character-rendering was directly inspired by classic terminal-based rogue-likes such as Rogue and Dwarf Fortress and also allowed for more anthropomorphism with the objects and classes without explicit definitions of their identities. These characters can be any symbol that can be represented on a terminal but lack the color variance found in some text-based roguelike games.

Because the system uses a finite-state machine as the basis of interaction and behavior, each node in the FSM graph represents a potential action state the entity object can be in. These actions are pre-programmed and define how an entity interacts with the environment. For this system, the possible action nodes are defined in Table I.

[4]Source code link removed for double-blind anonymization

Condition	Definition
none	*no condition is required to transition states*
step (int)	*every x number of simulation ticks the edge is activated and the node transitions*
within (char) (int)	*checks whether the entity is within a number of spaces from an instance of another entity with the target character*
nextTo (char)	*checks whether the entity is within one space (north, south, east, or west) of another entity of the target character*
touch (char)	*checks whether the entity is in the same space as another entity of the target character*

TABLE II: Entity FSM conditional edge definitions (ordered by least to greatest priority)



```
-- NODES --
0: idle
1: take
2: move
3: chase
-- EDGES --
0-1: touch $
1-0: none
0-2: step 2
2-0: none
0-3: within $ 5
```

Fig. 2: Example of both the text file output (left image) of the entity FSM representation and the translated visual graph representation (right image) for the character Link (L)

Each agent is initialized with the "idle" node as the starting base node, then a random subset of the possible action nodes is chosen from the simulation's provided configuration file to construct the rest of the graph. Each node can only be selected and added to the graph once to avoid redundancy and prevent over-complicated graph structures.

Edges of the agent's FSM definition are conditions that can occur within the simulation to allow node transitions of the agent from state to state. Like the nodes, these conditions are also pre-programmed. In the case that a node has multiple edges, the conditions are ordered by priority within the system and the edge with the highest conditional priority transitions to its endpoint state. For this system, the possible condition edges to connect the action-state nodes of the graphs are shown in Table II in the order of least to greatest priority.

The edges and their conditions are directional in relation to the nodes (i.e. an edge from node 0 to node 1 may have a different condition and definition than an edge from node 1 to node 0.) The entity can have from 0 to $n \times (n-1)$ edges. Like the node generation of the FSM, the edge connection and conditions are randomly generated from the subset of possible conditions provided in the simulation's configuration file.

At any time during the simulation, the entity is always in a state at one of the set nodes. At each timestep—or update within the fortress environment—each connection is evaluated to move to the connecting node state based on whether the conditions are met, in order of priority defined internally. The agent then performs the action at its new current node on the next timestep.

B. Fortress

The "fortress" class of the Amorphous Fortress contains the environment where the simulation takes place and stores general information accessible to all of the entities in the fortress. The borders and size of the fortress are defined initially with a set character width w and height h to enclose the entities. On initialization, the fortress generates each entity class FSM for each character defined in the passed configuration file. This global dictionary of entity classes allows any instance of an entity to add or transform different entity instances even if none currently exist on the map. The fortress maintains a list of currently active entity instances in the simulation and adds or removes them if called by ID value from an entity instance. The fortress also maintains positional data about each entity to return for conditional checks (i.e. whether a particular position has an entity located there.) A log of actions is maintained and keeps track of each action taken by each entity and the timesteps at which they occurred during simulation. This log is exported upon the termination of the simulation by the engine. The fortress has 3 termination functions that are checked by the engine defined as follows:

- *extinction*: checks whether there are no entities at all left in the simulation
- *overpopulation*: checks whether there exist more than $2(wh)$ entities in the simulation
- *inactivity*: checks whether an action has been taken by an entity in the last x timesteps

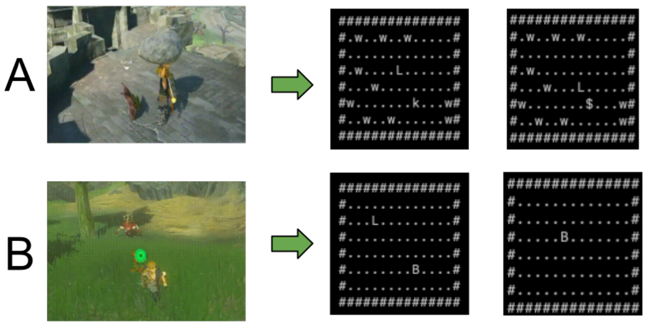
The cause for termination is added to the end of the log, along with every entity class definition's FSM tree, and labeled by the seed used to generate the simulation for reproducibility.

C. Engine and Main Loop

The engine serves as the entry point to the system and maintains the entire simulation as well as the update loop for the entities. The engine also imports the configuration file to pass to the fortress and to the entities - including the seed value for reproducible randomization. The engine randomly selects from the character class definition to create new instances to add the fortress at random positions. Once the entities and their FSMs are populated in the Fortress, the engine starts and updates the simulation. Each entity is evaluated at its current node and conditions are checked from the edges to move from that node onto a connecting node if possible. The engine continues updating the simulation until any of the termination conditions have been reached.

D. Example Scenario

To illustrate the potential of our system to generate plausible simulation dynamics, we use the FSM entity class definitions to manually design two simple example environments. These environments, inspired by the recent franchise entry *The Legend of Zelda: Tears of the Kingdom* [19], contains three entity class definitions: 'Link', 'Bokoblin' and 'Korok'. In the Zelda franchise, Koroks are characters normally revealed after the player, controlling the character Link, completes a puzzle or interacts with a unique hidden object, at which point the



```
###############
#.w..w..w.....#
#.............#
#.w....L......#
#...w.........#
#w.......k...w#
#..w..w......w#
###############
```

```
###############
#.w..w..w.....#
#.............#
#.w...........#
#...w...L.....#
#w.......$...w#
#..w..w......w#
###############
```

```
###############
#.............#
#...L.........#
#.............#
#.............#
#........B....#
#.............#
###############
```

```
###############
#.............#
#.............#
#.....B.......#
#.............#
#.............#
#.............#
###############
```

Fig. 3: Two example agent interactions from Zelda: Tears of the Kingdom remade in the Amorphous Fortress framework

Korok gifts the player a seed that can be used as in-game currency. Bokoblin characters in the game also interact with the character Link by chasing him when he comes into range and trying to attack him. Figure 3 shows a screenshot of both scenarios in game and recreated in the framework.

For scenario A, a single Link entity L and a single Korok entity k are initialized in the fortress with grass entities w placed as decoration. Link's FSM is defined such that the character will *move* randomly on the map every *2 steps*. When the Korok ('k') is *next to* Link ('L'), it *transforms* into a seed, '$'. When Link *touches* the seed, he *takes* it, removing the '$' character from the map. For scenario B, a Bokoblin entity B will *move* randomly in the fortress. If Link is *within* 5 spaces of the Bokoblin, the Bokoblin will *chase* him. If the Bokoblin is *touching* Link, Link will *die*.

IV. Experiment 1: Observing Emergent Behaviors

To explore the potential depths of the generated finite state machine behaviors, we implement a hillclimbing evolutionary algorithm to evolve the agents and fortress towards having the largest, but also most traversed FSMs.

A. Experiment Setup

Three different mutation algorithms were implemented for evolution. The first involved mutating the nodes of randomly selected entity classes. This was done by either 1) deleting random nodes (so long as there were more than one node in the graph) 2) adding more nodes (if there were any that have not been added already) or 3) replacing nodes with another action. Similarly, the second mutation modified the edges by either 1) deleting random edges (so long as there was more than one edge in the graph), 2) adding edges between nodes, or 3) overwriting the conditional check on an edge. The third mutation modified the number of instances in the fortress by adding or removing instances. After mutation, the modified FSM was pruned up for any unconnected edges and orphaned nodes after mutation of the finite state machines. The mutation was done by coin-flip chance, given some preset probability for both the node level, edge level, and instance level mutation. Algorithm 1 demonstrates the mutation process for the evolutionary system of the fortress. The mutations of this algorithm were applied at the entity's class level definition - such that each future instance of an entity class would all have the same modified FSM tree definition. The number of instances initialized in the fortress were also mutated from generation to generation. We chose to use the hillclimber evolutionary algorithm for this study as we were only concerned with exploring how the fortress and simulation could gradually evolve toward an objective as a whole. The hillclimber algorithm for this experiment also allowed for the simplest open-ended emergence for novel agent behaviors.

Algorithm 1: Mutation function for the Fortress

```
1  $node\_prob$, $edge\_prob$, $instance\_prob$
2  $node\_r$ = random();
3  $edge\_r$ = random();
4  $instance\_r$ = random();
   /* Mutate random entity class nodes */
5  while $node\_r < node\_prob$ do
6      $i$ = random(0,2);
7      $e$ = random($fortress.ent\_def$);
8      if $i == 0$ then
9          $fortress.\_delete\_node(e)$;
10     else if $i == 1$ then
11         $fortress.\_add\_node(e)$;
12     else if $i == 2$ then
13         $fortress.\_alter\_node(e)$;
14     $node\_r$ = random();
   /* Mutate random entity class edges */
15 while $edge\_r < edge\_prob$ do
16     $i$ = random(0,2);
17     $e$ = random($fortress.ent\_def$);
18     if $i == 0$ then
19         $fortress.\_delete\_edge(e)$;
20     else if $i == 1$ then
21         $fortress.\_add\_edge(e)$;
22     else if $i == 2$ then
23         $fortress.\_alter\_edge(e)$;
24     $edge\_r$ = random();
   /* Mutate random entity instances in the
      fortress */
25 while $instance\_r < instance\_prob$ do
26     $i$ = random(0,1);
27     $e$ = random($fortress.entities$);
28     if $i == 0$ then
29         $fortress.\_remove\_entity(e)$;
30     else if $i == 1$ then
31         $x, y$ = random(fortress.pos);
32         $fortress.\_add\_entity(e, x, y)$;
33     $instance\_r$ = random();
```

In this experiment, we wanted to encourage more diverse and deep behaviors from the generated FSMs of the fortress. Therefore, our fitness function was based on if a node or edge of an FSM was traversed during the entire course of the simulation. The fitness function, f, for evolving the fortresses using the hill-climbing algorithm was defined as the following equation: $f = \frac{v}{u+1} \times t$ where v is the sum of the number of traversed (or visited) nodes and edges of every FSM in the fortress, u is the combined number of unused (or unvisited) nodes and edges of each FSM, and t as the total number of nodes and edges in the entity class FSMs. The visitation of a node or edge in an FSM was determined by the combined behaviors of every instance of a particular entity. Because some instances of an FSM may have different experiences and interactions based on initial placement, we used the combined behaviors of all instances of an entity class presented in the fortress for the fitness function. For example, if a single instance of the '@' class completes an action that is part of the FSM's tree due to a conditional check successfully occurring, both the node it traveled to and the edge condition it traveled by would be considered 'visited' for the entire entity class of '@'. This fitness function was also defined to encourage both the generation of large but thoroughly explored FSMs for the fortress overall and not just by a single entity class.

For this study, we examine the evolution of 5 fortresses each starting from randomly generated seeds. We evolved the fortress for a total of 1000 generations – comparing the fitness of one new mutated fortress to the best-generated fortress. If the generated fortress achieved a higher fitness score, the best fortress would be replaced by the current instance, and so on. The fortress was simulated for 20 timesteps before being evaluated for entity tree traversals. This was to limit the computational calculations of each instance in the fortress while still allowing for depth and interaction with the agent behaviors. This study used a 50% independent probability for all mutation chances described before – a 50% chance to mutate a randomly selected entity class's FSM nodes, edges, and/or to add or remove an instance from the fortress. A total of 15 unique ASCII characters were used for the experiments, and all edge conditions and node action conditions were allowed for selection.

B. Results

We observed a number of relatively interesting behaviors from both the evolutionary experiment, the final entity FSM definitions, the fortress construction, and the trends of the fortress evolution. Figure 4 shows the fitness score composition over the 1000 generations over the 5 trials, which steadily improves over time. The following subsections describe these phenomena in more detail along with our insights and hypothesis for these emergent behaviors.

1) FSM Objective Coverage: A large majority of the entity classes in the final best fortresses had high graph traversal coverage overall. The calculated average across all 5 fortresses for every entity class had an average of 72% node coverage and 58% edge coverage. The function was designed with the intention to reward large, well-traversed finite state machine graphs - while giving less incentive to generate large graphs with less traversal and small graphs with fuller traversal. While many generated entity classes had large graphs - defined as

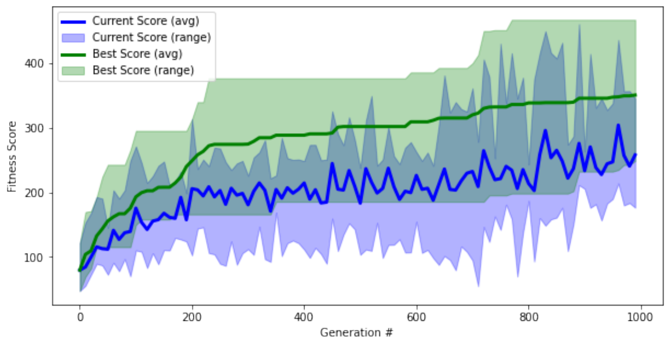
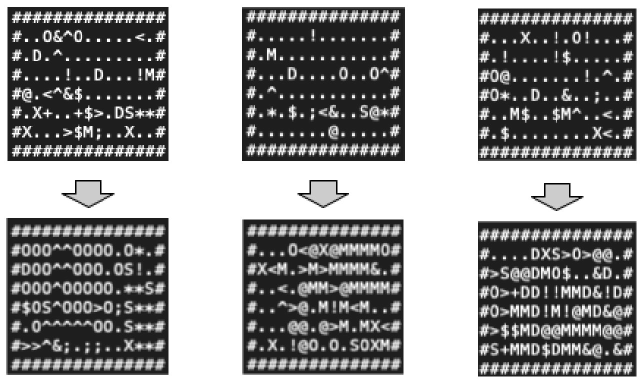


Fig. 4: Best and current fitness scores of the 5 experiment trials (averaged and min/max range

having 7 or 8 nodes and between a range of 15-30 edges - and strong node/edge coverage, there was also a small number of entities that had significantly smaller graphs - with 2 or fewer nodes and less than 5 edges. This diversity of entities regardless of the size of the graphs may have been due to a naturally emerging "symbiosis" of the entities and their relationships to each. For example, although one entity class may not have a more actively influential node such as *take* or *add*, the conditional edges of another entity's class could have depended on the presence of that entity in order to do something more influential — i.e. another entity may have needed to be *next to* or *touch*ing the small-graph entity in order to *clone* itself or *transform*.



```
################
#..0&^0.....<.#
#.D.^.........#
#....!..D...!M#
#@.<^&$.......#
#.X+..+$>.DS**#
#X...>$M;..X..#
################
```

```
################
#.....!.......#
#.M...........#
#...D....0..0^#
#.^...........#
#.*.$.;<&..S@*#
#.......@.....#
################
```

```
################
#...X..!.0!...#
#.!....!$.....#
#0@.......!.^.#
#0*..D..&..;..#
#..M$..$M^..<.#
#.$........X<.#
################
```

```
################
#000^^0000.0*.#
#D00^^000.0S!.#
#000^00000.**S#
#$0S^000>0;S**#
#.0^^^^^00.S**#
#>>^&;.;;..X**#
################
```

```
################
#...0<@X@MMMM0#
#X<M.>M>MMMM&.#
#..<.@MM>@MMMM#
#..^>@.M!M<M..#
#...@@.@>M.MX<#
#.X.!@0.0.S0XM#
################
```

```
################
#....DXS>0>@@.#
#>S@@DM0$..&D.#
#0>+DD!!MMD&!D#
#0>MMD!M!@MD&@#
#>$$MD@@MMMM@@#
#S+MMD$DMM&@.&#
################
```

Fig. 5: Initial and final renderings of the fortress for 3 seeds

2) Rainbow Goop FSM Generation: Because of the objective function, the FSM shifted towards having "reproductive" behaviors. The entities evolved towards focusing on having node actions that created more of themselves but also changed themselves into another entity. As such, the *add*, *transform*, and *clone* nodes were seen more frequently in the entities graphs than more destructive nodes such as *take* and *die*. Based on the initial and final map renderings, many of the initial entities on the map would try to clone itself to create more copies of itself — this allowed for deeper graph exploration to fulfill the "visited" over "unvisited" node statistic. Afterward, many entities would often *transform* into other entities — to fulfill other missing nodes and edge visitations concerning other entities that may not have initially been on the map. This explains the copious — almost overpopulated — amount of entities in the final rendering of the fortress simulation. Figure 5 shows the initial fortress renderings and the final fortress renderings after 20 steps of simulation. More diversity of characters and entities on the map also encouraged more interactions overall. We call this type of behavior "rainbow goop" as the behavior replicates itself, but also tries to transform itself into other entities to diversify the fortress.

V. Experiment 2: Finding Diverse Environments

Following the trends of the last experiment, we wanted to see if it was possible to generate multiple fortresses capable of varied emergent behavior, similar to that of the rainbow goop and symbiotic relationship. We achieve this by optimizing both quality and diversity of the generation process using metrics reflecting agent behavior and interaction in the fortress environments.

A. MAP-Elites for Amorphous Fortress

We implement the MAP-Elites QD algorithm [15] to evolve multiple fortresses towards a diversity of emergent behaviors. The emergent behaviors of the entities defined within this system are dependent on interactions with other instances within the same fortress. Therefore a single cell of the MAP-Elites grid contains a fortress with its own set of entity class FSM definitions. Figure 7 illustrates a small example of the evolution and evaluation process as the fortresses are placed in the MAP-Elites grid for the experiment (described in more detail later in this section).

1) Behavior Characteristics: For a MAP-Elites implementation, the dimensions of the archived QD grid are known as the behavior characteristics (BCs). These BCs designate how the individuals from a population are separated and maintained for sample diversity and replaced within the cell to improve quality. We define the following behavior characteristics that are used for experiment: a) the mean number of total instances in the fortress at the end of the simulation (population) and b) the mean number of total nodes across all entity class definitions (total FSM size.)

For behavior characteristic a based on the number of entity instances, this dimension is intended to explore the population of a fortress, whether the entity class combinations result in an overpopulation of entity instances, an extinction of all instances, or a stability or "equilibrium" of instances within the fortress. The values of this dimension can range from 0 to 156—the maximum number of instances allowed to exist in the fortress before it terminates based on an "overpopulation" condition.

Behavior characteristic b, based on the collective class FSM size, looks to examine the "complexity" and "depth" of the entity classes; whether the combined set includes a majority of simple entity class definitions with only 1 action node or conversely with extremely large entity class definitions. The

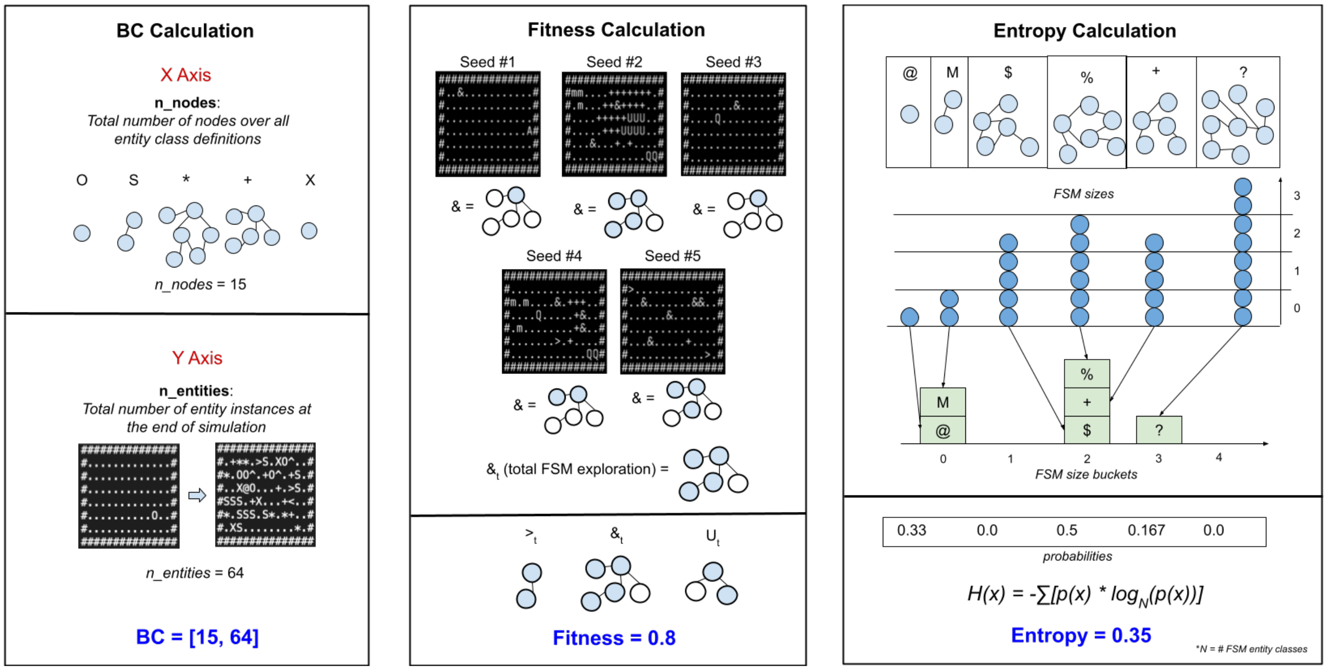


Fig. 6: Diagram of the calculations for MAP-Elites BCs, fitness values, and entropy value

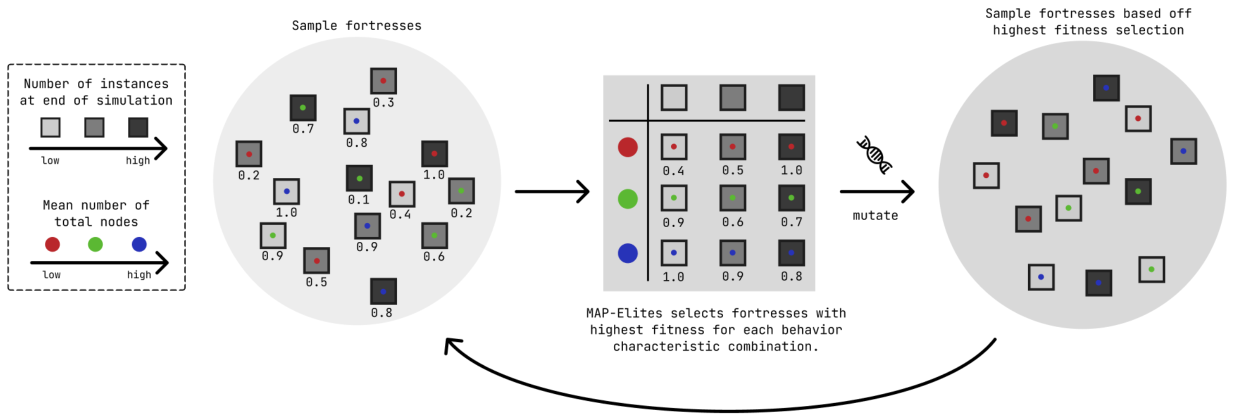

Number of instances at end of simulation
low
high
Mean number of total nodes
low
high
Sample fortresses
MAP-Elites selects fortresses with highest fitness for each behavior characteristic combination.
mutate
Sample fortresses based off highest fitness selection

Fig. 7: Diagram of the MAP-Elites algorithm applied to the Amorphous Fortress QD experiment

values of this dimension can range from 15 nodes—where each of the entity classes has only 1 node—to 1400 nodes—where each possible node is included every entity class FSM definition. The exploration of this dimension by the MAP-Elites algorithm will demonstrate the growth and utility of varying sized entity class FSMs.

2) *Population:* An evaluated population consists of a fortress with a set of agent entity class definitions. The population size for this experiment was 10 individuals per generation—9 sampled from elites and 1 randomly created similar to the initialization. The singular random fortress is injected into the set of mutant elites to encourage exploration within the MAP-Elites grid and to prevent the algorithm from reaching a local minimum during evolution. We parallelize the evaluation of these 10 individuals to speed up evolution. On initialization, all 15 of the FSMs are defined for each fortress individual. In this step, the number of total nodes to be added over all FSMs in the fortress is sampled uniformly to encourage a larger spread of randomly initialized individuals over the MAP-Elites grid (along the n. nodes axis), thus allowing for more uniform exploration of cells as evolution progresses.

We randomly initialize fortresses so as to sample uniformly along the axis measuring the number of aggregate FSM nodes. We first uniformly sample this aggregate number, then split it into as many summands as there are entity types using an evenly weighted multinomial distribution, where each summand corresponds to the number of nodes to be assigned a given entity. It is possible in this setting for an entity type to be assigned more nodes than there are distinct node types; in this case, we (greedily) re-assign the surplus nodes to one or more non-overfilled entity classes, until no surplus nodes remain.

3) Mutation: A fortress individual in the population is mutated by modifying its genotype: the class level definitions of the FSMs. Each fortress contains 15 entity classes, where each class can have a minimum of 1 action node and a maximum of 95 action nodes. This process is done similarly to the first experiment. However, as a modification for the MAP-Elites experiment, the node mutation is adjusted to increase exploration within the grid. Unlike the previous experiment where a single node was modified per coin-flip chance, a range of nodes can be added, removed, or altered into another action node definition. For example, 10 nodes can be added to one entity class definition, while 4 are removed from another (or the same if randomly chosen again).

4) Fitness: The fortress sample individuals are evaluated based on the ending state of the fortress. The fortress is simulated for 100 steps, where each instance of an entity class present in the fortress enacts the current action node of its FSM graph once per step and then evaluates the next action node to move to based on the state conditions it ends in. The fitness function of the MAP-Elites implementation of the system is similar to the hillclimber experiment. This fitness definition encourages each class entity to have the full possibility of its emergent behaviors demonstrated within the simulation. The final exploration of a class definition's nodes and edges are also aggregated over evaluation trials in case different behaviors occur due to different seed evaluations. The fitness function for a fortress is defined with the following equation: $f = e/t$ where f is the fitness value from 0 to 1, e is the total number of explored nodes—nodes that were activated during simulation—for all entity class definitions in the fortress individual and t is the total number of nodes—activated or un-activated—for each entity class in the fortress individual.

5) Entropy of the FSM definitions: Entropy examines the distribution of the sizes across the entity class definitions. The values of this dimension ranges from 0 to 1, with 0 meaning all of the entity class FSMs have the same number of nodes and no variation, and 1 meaning the number of nodes are different for each entity class FSM definition. We use Shannon Entropy to calculate the entropic value of the FSM sizes with a b base N where N is the number of FSM size bins (which for this experiment is equal to the number of entity classes).

Figure 6 shows how the BCs, fitness value, and entropy value are calculated for any given fortress.

B. Experiment setup

Using the fitness function, Amorphous Fortress (AF) mutation operators, and AF initialization schemes described above, we use standard MAP-Elites to iteration on a population of AF samples, evaluating them to determine fitness measures and characteristics, sorting them into their respective MAP-Elites grid cells, re-sampling from the grid to create the new population of samples and repeating for a set number of generations. In this experiment, we use a population sample size of 10 fortresses each with 15 entity class definitions, evaluate these fortresses across 10 randomly chosen seeds, and evolve the populations for 10,000 generations. We use a small number of samples for the population with significantly longer generation time to encourage the algorithm to explore more of the cells of the MAP-Elites grid. The initial population of fortresses is made up of FSM definitions with only a single uniformly random action node. We use two combinations of behavior characteristics in this experiment to explore the QD space of the evolved fortresses as well as the emergent behaviors from the elite fortresses saved.

C. Results

1) Map Coverage: Figure 8 shows the heatmap for the MAP-Elites grid with the fitness for this experiment measuring the percentage of the entity class FSM visited. Nearly every possible MAP-Elites grid cell was filled for this experiment. Fortresses with fewer total nodes had the highest fitness values in the grid—most likely because it becomes increasingly challenging to explore different nodes in larger graphs. Predictably, many cells that contained fortresses with high instance numbers were terminated from overpopulation. The number of total entity instances does not seem to be limited by the total number of nodes, with fortress environments that lead to either extremes of near extinction (0-1 instances) or overpopulation (156 instances.)

After the QD search process, we additionally evaluate the diversity of the entity classes within each fortress, measured as the entropy over the distribution of entity class FSM sizes. We note that high entropy—exhibited in a large swatch of the network with a medium-high number of nodes—corresponds to sets of entities with variably sizes FSMs. When such individuals are fit (and FSMs have few ineffective nodes/edges), we can guarantee that different types of entities will exhibit diverse behavior. In this case, a hypothetical learning agent will be forced to adapt to a diversity of behavior profiles, increasing the richness of its task.

2) Fortress Population Equilibrium: We observed an interesting phenomena within the MAP-Elites cells concerning the population numbers of each of the entities. Ideally, we were aiming to find fortresses with balanced interactions between entities. Figure 9 shows the growth of the population in the fortresses found in the ending axis of the MAP-Elites archive. The cells found at the extreme points of the archive (i.e. cells with lowest and highest possible behavior characteristics) exhibited uncooperative behavior between the instances. Fortresses with lower instance numbers refused to populate and caused a stagnation in the fortress. Conversely, fortresses with higher instance numbers quickly overpopulated. However, fortress individuals found in the middle of the archive had more "equilibrium" and cooperation. The fortresses in the middle of the archive achieved much more diversity in terms of the entity class population; some entity classes having a sudden growth in instance number before dying off, while others slowly expanded their presence over time. This "equilibrium" was most noticeable in the fortress individual that demonstrated the highest entropy between entity class FSM

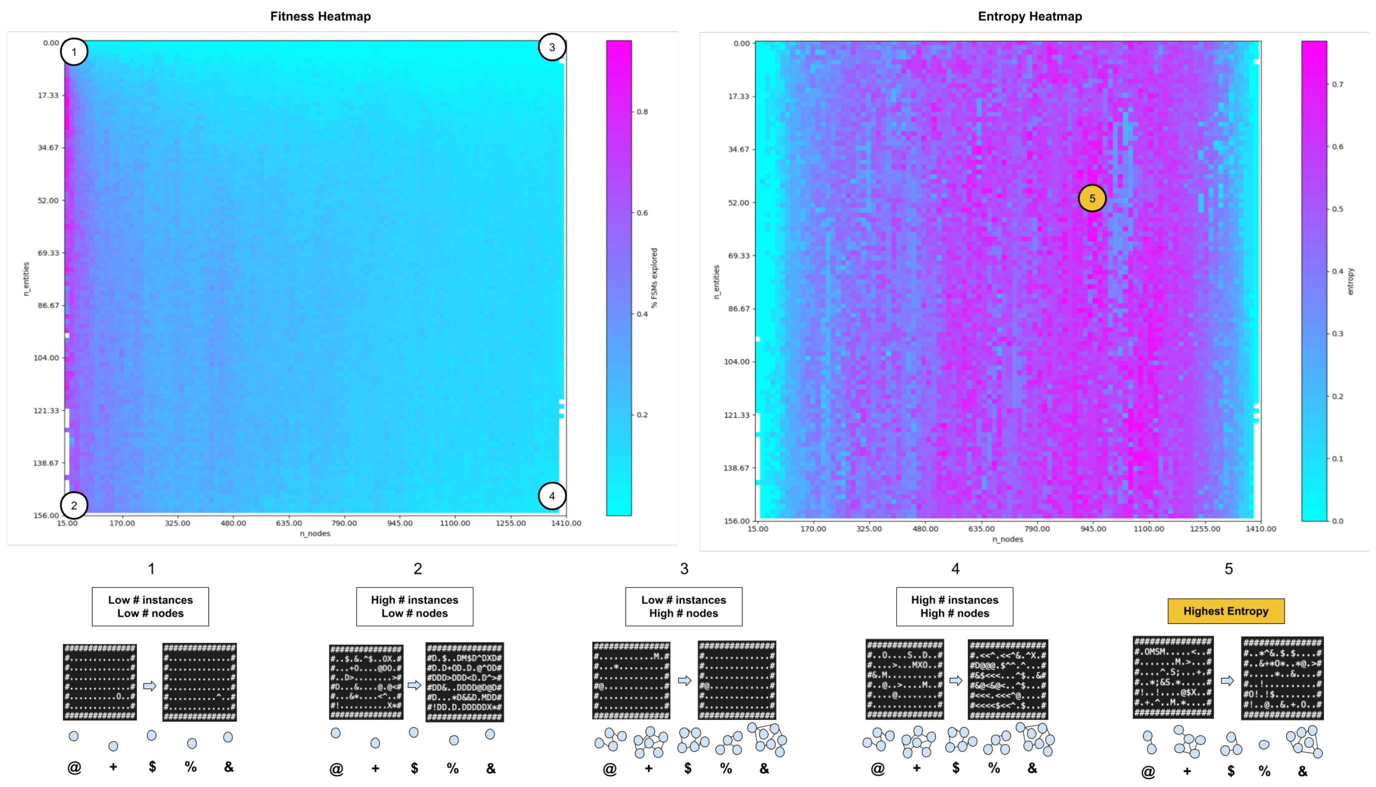


Fig. 8: Results of the AF-QD experiment: exploring **number of instances** at the end of the simulation vs. **total number of nodes**. The graph on the left shows the heatmap with respect to the fitness measurement—the proportion of FSMs explored in the fortresses. The graph on the right shows the same archive of fortresses measured with the heatmap showing the measure of FSM entropy sizes.

sizes. This fortress showed a near perfect balance between all entity classes; neither dominating nor diminishing in numbers. The entities found in this fortress find a "harmony" of co-existence where the ecosystem does not find itself in danger of overpopulation nor extinction. From this, we can conclude that having a diversity of entity class sizes leads to better balance of entity populations and allows for more exploration of co-operative class behaviors.

3) Limitations: The main weakness of our results is the generally low fitness, which indicates that much of the larger FSMs generated by our system could be pruned to drastically smaller size without having any effect on environment dynamics. We hypothesize that this lack of FSM exploration is the result of limited compute resources. In particular, 100 steps of simulation is not likely enough to explore FSMs with up to 94 nodes. Conversely, the small map size of generated environments may make prohibit more interesting large-scale dynamics. We observe the QD score to still be rising steadily after 10k iterations, such that further evolution would be beneficial. Our QD search experiments, and subsequent re-evaluation on new seeds for more episode steps, emphasize environments that maintain equilibriums between entity types over long time horizons; thus, showing how to optimize for environments facilitating novel dynamics for learning agents.

VI. Future Work

The results from these experiments highlight the benefits and limitations of the AF framework and how we can better explore the potential of the system for future open-ended AI research. For the QD experiment in particular, we would like to explore the effects of another behavior characteristic that is more dependent at an entity class definition level. For example, one archive dimension could measure how many times the "take" node is enacted by entities could encourage the evolution of fortresses with more or less aggressive entities. We would also like to examine the compressibility (e.g. via a simple gzip algorithm) or predictability (e.g. by a a neural network trained with supervised learning) of environment rollouts generated by a given fortress definition. We expect that such measures will provide a reasonable estimate of a hypothetical learning agent's ability to model and/or adapt its behavior to a given fortress [20], and could thus be used to validate the effectiveness of our FSM-based complexity metrics (themselves being cheaper to compute) and/or supplant them (if necessary).

From the engineering side, the fortress engine could likely be drastically accelerated if it was implemented in a batched, GPU-compatible manner, similar to the recent trend in RL environments which has allowed for orders of magnitude

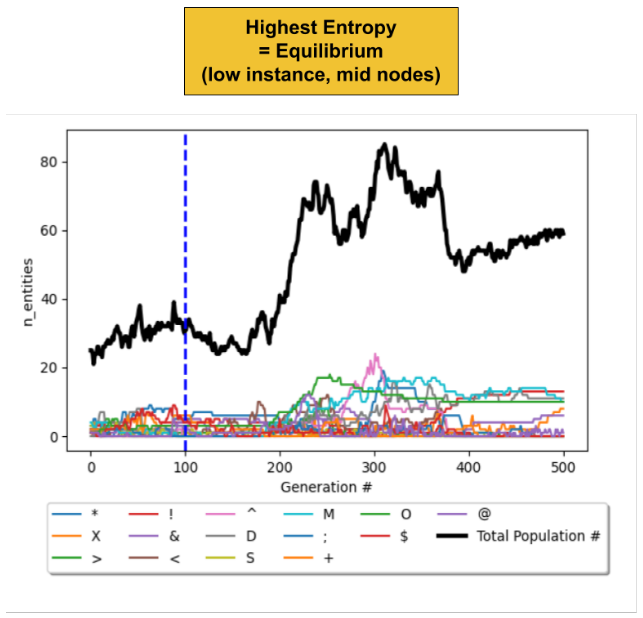


Fig. 9: **Growth of the population** number for each entity class in the MAP-Elites archive. The cell with the highest entropy of entity class definitions achieves an ecological equilibrium. This behavior is distinct from the extremes of the archive, which either overpopulate or rarely add any additional entities.

increases in simulation speed [21], [22].

A different line of future work—emphasizing the QD-AF paradigm as a design tool in its own right—could involve developing a mixed-initiative online system in which users are free to design their own fortresses and entity class definitions. The MAP-Elites fortress illumination process could create a "casts" of generated characters for the user to include as a recommendation engine. These generated entity classes would highlight and enhance the potential behaviors singular "main character" entity within the fortress. It would be interesting to explore the potential of the collaborative nature between the users and the system to generate distinct entities, specific to the user's preference. Additionally, users would be able to share and modify submitted fortresses to create "stories" using shared entity characters.

VII. Conclusion

We introduce the Amorphous Fortress framework – a microscopic and fully transparent zero-player simulation environment made up of finite state machine agents. We explore the potential of the system through two evolutionary experiments: a hillclimber experiment intended to generate emergent behaviors from the multi-agent interaction and a quality diversity experiment intended to find variety in agent behavior across environments. This open-source framework lays the foundation for more research focusing on explainable AI, multi-agent learning, and open-ended simulation studies.

References

[1] M. Gardner, "Mathematical games," *Scientific american*, vol. 222, no. 6, pp. 132–140, 1970.

[2] J. Orkin, "Three states and a plan: the ai of fear," in *Game developers conference*, vol. 2006. CMP Game Group SanJose, California, 2006, p. 4.

[3] I. Y. Smolyakov and S. A. Belyaev, "Design of the software architecture for starcraft video game on the basis of finite state machines," in *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 2019, pp. 356–359.

[4] H. Zhou, Y. Gong, L. Mugrai, A. Khalifa, A. Nealen, and J. Togelius, "A hybrid search agent in pommerman," in *Proceedings of the 13th international conference on the foundations of digital games*, 2018, pp. 1–4.

[5] D. Olsen and M. Mateas, "Beep! beep! boom! towards a planning model of coyote and road runner cartoons," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, 2009, pp. 145–152.

[6] M. Charity, D. Rajesh, R. Ombok, and L. B. Soros, "Say "sul sul!" to simsim, a sims-inspired platform for sandbox game ai," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 182–188.

[7] S. Earle, "Using fractal neural networks to play simcity 1 and conway's game of life at variable scales," *arXiv preprint arXiv:2002.03896*, 2020.

[8] M. C. Green, V. Yen, S. Earle, D. Rajesh, M. Edwards, and L. B. Soros, "Exploring open-ended gameplay features with micro rollercoaster tycoon," *arXiv preprint arXiv:2105.04342*, 2021.

[9] M. Kreminski, ""generator's haunted": A brief, spooky account of hauntological effects in the player experience of procedural generation," in *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, pp. 1–3.

[10] F. S. Nicholls and M. Cook, "'that darned sandstorm': A study of procedural generation through archaeological storytelling," *arXiv preprint arXiv:2304.08293*, 2023.

[11] P. R. Norvig and S. A. Intelligence, "A modern approach," *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, vol. 90, pp. 33–48, 2002.

[12] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty." in *ALIFE*, 2008, pp. 329–336.

[13] M. Beukman, C. W. Cleghorn, and S. James, "Procedural content generation using neuroevolution and novelty search for diverse video game levels," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1028–1037.

[14] A. S. Melotti and C. H. V. de Moraes, "Evolving roguelike dungeons with deluged novelty search local competition," *IEEE Transactions on Games*, vol. 11, no. 2, pp. 173–182, 2018.

[15] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[16] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering quality diversity in dungeon design with interactive constrained map-elites," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[17] C. Guerrero-Romero and D. Perez-Liebana, "Map-elites to generate a team of agents that elicits diverse automated gameplay," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.

[18] E. S. Norstein, K. O. Ellefsen, and K. Glette, "Open-ended search for environments and adapted agents using map-elites," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2022, pp. 651–666.

[19] Nintendo, "The legend of zelda: Tears of the kingdom," [Nintendo Switch], 2023.

[20] F. J. Gomez, J. Togelius, and J. Schmidhuber, "Measuring and optimizing behavioral complexity for evolutionary reinforcement learning," in *International Conference on Artificial Neural Networks*. Springer, 2009, pp. 765–774.

[21] R. T. Lange, "evosax: Jax-based evolution strategies," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 659–662.

[22] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax-a differentiable physics engine for large scale rigid body simulation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.