Say “Sul Sul![*]” to SimSim, A Sims-Inspired Platform for Sandbox Game AI

Megan Charity
Tandon School of Engineering
New York University
mlc761@nyu.edu

Dipika Rajesh
Tandon School of Engineering
New York University
dr2898@nyu.edu

Rachel Ombok
Tandon School of Engineering
New York University
rbo224@nyu.edu

L. B. Soros
Tandon School of Engineering
New York University
lsoros@nyu.edu

Abstract

This paper proposes environment design in the life simulation game The Sims as a novel platform and challenge for testing divergent search algorithms. In this domain, which includes a minimal viability criterion, the goal is to furnish a house with objects that satisfy the physical needs of a simulated agent. Importantly, the large number of objects available to the player (whether human or automated) affords a wide variety of solutions to the underlying design problem. Empirical studies in a novel open source simulator called SimSim investigate the ability of novelty-based evolutionary algorithms to effectively generate viable environment designs.

Introduction

Sandbox games require constrained creativity and exploring complex design spaces while avoiding nonviable portions. Examples of such games include Minecraft[1], in which players must build shelters out of natural resources, and SimCity[2], which requires players to set zoning policies and otherwise allocate resources to enable simulated city population growth. This genre is unique because success can be achieved using an enormous variety of strategies. While this relative freedom creates a unique gaming experience for human players, it also presents an underexplored challenge for game AI: finding a diversity of creative artifacts that satisfy a set of minimal constraints. Such domains can serve as an apt testbed for an underexplored class of divergent evolutionary search algorithms that are driven primarily by binary minimal fitness criteria instead of continuous fitness gradients.

In The Sims ((c) 2008 Electronic Arts, Inc.), the player controls simulated humans called Sims that inhabit a house, which can be either preconstructed or built and furnished by the player. An example human-built house is shown in Figure 1. Each Sim has a set of needs (Hunger, Energy, Social, Fun, Bladder, and Hygiene) that must be satisfied, with satisfaction levels decreasing over time. Interactions with objects, such as furniture, and with other Sims can have positive and negative effects on need satisfaction levels. The primary player task is then to select actions that will prevent the Sim's needs from becoming critically low. Importantly, there are no “win” conditions beyond any personal goals that the player might set. However, a Sim can die if certain needs go unmet for too long. Thus, The Sims contains two separate yet related gameplay challenges: 1) designing a house with sufficient objects to prevent a Sim's needs from becoming critically low, and 2) selecting interactions with these objects such that the Sim's decaying need satisfaction levels are sufficiently replenished.

This paper introduces a new open source simulator based on house design in The Sims and demonstrates its utility for exploring minimal-criterion-based search algorithms. Though prior work by e.g. Yu et al. (2011) has explored optimizing furniture arrangement for realistic living spaces, the goal herein is to discover quality diversity in the context of minimal viability criteria. The new experimental domain and accompanying source code are the primary contributions of this paper, in addition to arguing more generally that sandbox games are interesting testbeds for AI and, more specifically, for divergent search algorithms. First, existing work on this class of algorithms is reviewed. The SimSim Sims simulator is then introduced and key experiment-enabling differences from the original game are noted. Experiments with variants of an algorithm called minimal criterion novelty search are then reported, highlighting the novel platform's utility for future studies on divergent evolutionary search algorithms and procedural content generation.

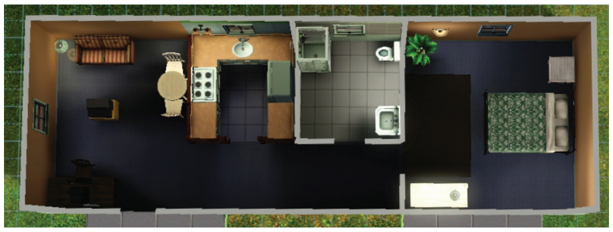


Figure 1: A human-designed house in The Sims 3. The SimSim simulator simplifies the design problem into that of designing a single functional room (with each furniture object occuping a single tile), though the simulator also supports evolving entire houses.

[*]*Sul sul* means *hello* in Simlish, the fictional language spoken in The Sims.

[1]Copyright (c) 2011 Mojang

[2]Copyright (c) 1989 Electronic Arts, Inc

Background

Evolutionary computation (EC) is a class of biologically-inspired algorithms encoding points in a search space as genomes (i.e. mutatable parameter vectors). While EC methods differ in terms of genome representation (and potentially other features), they all share a common approach of exploring a search space by mutating and sometimes combining genomes representing already-discovered individuals. The general idea, motivated by the concept of "survival of the fittest", is to iteratively create a population of genomes, score them on some fitness-bearing domain, and then select the elites as parents for the next generation.

? (**?**) turned this idea on its head, demonstrating that novelty alone, without fitness-based selection, could solve a maze navigation problem not solvable by fitness-based search alone. This idea was later extended by minimal criteria novelty search (MCNS) (**?**), which increases search efficiency by requiring that individuals satisfy some minimal fitness threshold in order to reproduce. This algorithm was transformative because it emphasized "survival of the fittest" instead of "survival of the fit *enough*", allowing more room for evolutionary creativity. Further innovations on MCNS include progressive minimal criterion novelty search (Gomes, Urbano, and Christensen 2012), evolution in Chromaria (**?**), minimal criterion coevolution (Brant and Stanley 2017), and POET (Wang et al. 2019).

Novelty search, in part inspired a recently popular class of evolutionary algorithms called quality diversity (QD) algorithms that aim to discover a wide variety of high-performing genomes within a particular search space. Most QD algorithms derive from either novelty search with local competition (NSLC)(**?**) or MAP-Elites (**?**). QD algorithms have also been used for procedurally generated game content such as spaceships (Liapis et al. 2013), strategy game maps (Liapis, Yannakakis, and Togelius 2015), bullet hell game levels (Khalifa et al. 2018), and RPG dungeons (Alvarez et al. 2019). Gravina et al. (2019) give a review of QD+PCG work.

Though intentionally divergent, QD algorithms are frequently cast as optimizers. More critically, the domains used to evaluate them rarely capture the prolific creative potential of evolutionary generative systems. Initial experiments focused on single-objective maze navigation tasks. Subsequent maze domains involved more complicated mazes explicitly designed with multiple viable paths to the end point (**?**), shifting the goal to finding *all* such paths, not just one. Artificial life domains for QD have largely focused on morphologies or gaits for virtual creatures or physical robots. Games such as Hearthstone (Fontaine et al. 2019) have also been used as testbeds for QD algorithms. Still, most of these domains naturally implement a gradient-based fitness function, enabling "survival of the fittest" instead of "survival of the fit enough".

The novel sandbox-game-based simulator described in the next section implements a natural minimal fitness criterion, creating an important new testbed well-suited to this important new class of algorithms. A few prior Other works have posed sandbox games as challenges for AI in general. For instance, Earle (2020) explored reinforcement learning in SimCity. Minecraft has become an increasingly popular domain for creative AI research in games. In the Generative Design in Minecraft (GDMC) competition (Salge et al. 2020), the primary challenge is for an algorithm to generate plausible settlements on arbitrary terrains. Submissions are scored by human judges with respect to adaptation, functionality, evocative narrative, and aesthetics. Importantly, while the theme of generative design is shared between GDMC and SimSim, and while GDMC implicitly rewards algorithms that can generate a diversity of high-quality designs (by evaluating submitted algorithms on multiple terrains), GDMC does not incorporate the survival aspect of Minecraft and thus is not an ideal testbed for minimal-criterion-based algorithms. Inversely, the MineRL competition[3] focuses on high-performing agent gameplay behaviors, with little emphasis on creativity. **?** (**?**) introduced a simplified Minecraft domain called Voxelbuild explicitly designed for evaluating divergent evolutionary search algorithms. However, this system removes the Minecraft survival mechanic, thereby removing the minimal viability criterion from the game.

SimSim: A Sims Simulator

The SimSim simulation[4] replicates select game mechanics from The Sims for the purpose of exploring algorithms that find a diversity of high-quality designs with minimal viability criteria. This section first describes game mechanics in The Sims before noting how SimSim is different.

In The Sims, a human player commands one or more Sim agents to navigate to particular objects and interact with them, replenishing one or more needs. Sims can also select their own goals probabilistically; when not controlled by the player, Sims use A* pathfinding to move towards one of the four highest-ranked objects according to impact on "happiness" (combined Sim needs ordered by Maslow's Hierarchy of Needs). This decision-making process can also be influenced by distance to objects, personality motives, interaction with other Sims, and mood (Bourse 2012).

In SimSim, a single agent (Algorithm 1) lives in one-room houses and interacts with furniture objects. The simulator contains 70 hand-coded objects, each with a vector of effects on Sim needs. For the experiments in this paper, objects are placed by evolutionary algorithms, but other strategies could be implemented. There is no player interaction, and the Sim has full knowledge of every object in the room. Needs are prioritized based on a preset order, so that the Sim will attend to a more critical need (i.e. hunger or energy) over others. (SimSims die if Hunger or Energy reaches 0.) Navigation uses breadth-first search in a room with X-Y positive integer coordinate values. The agent can take one step up, down, left, or right each tick. Once the agent reaches its target object, it can interact with it. SimSims interact with objects until the relevant diminished need is satisfied. Some objects replenish a need more than others per interaction, but it is unlikely for all needs to be maximally satisfied. SimSim only allows one agent at a time and social interactions are not implemented beyond some objects satisfying the Social

[3]https://minerl.io/competition/

[4]Code: https://github.com/lsoros/simsim.

need. SimSim agents also only have one possible generic interaction with each object, instead of selecting from options that might each affect the Sim's needs differently. SimSim happiness is a simple combination of all needs, whereas happiness in The Sims is impacted more strongly by unfulfilled needs than fulfilled ones.

Algorithm 1: SimSim Agent

```
Input: maxTicks, dimRate, needsRankings, thresh
Output: fitness
1  while $curTick < maxTicks$ do
       /* Check if Sim agent is dead    */
2      if $hunger \leq 0$ or $energy \leq 0$ then
3          break
       /* Set target path if not yet set
           */
4      if ($\exists targetObject$) and ($\neg\exists targetPath$) then
5          $targetPath$ = bfs($targetObject$)
       /* Move to target and interact   */
6      goto next path point
7      if sim at $targetObject$ position then
8          Interact with $targetObject$
9          $targetObject$ = null
       /* Diminish needs based on tick  */
10     for $i$ in (# sim needs) do
11         if (dimRate[$i$] % $curTick$) = 0 then
12             simNeeds[$i$] -= 1
       /* Set the next target object    */
13     if $\exists targetObject$ then
14         continue
15     for $need$ in $needsRanking$ do
16         if simNeeds[$need$] < $thresh$ then
17             $targetObj$ = findNearestObj($need$)
18     $curTick$++
19 return $fitness$
```

Methodology

This paper explores the impacts of varying the fitness threshold for minimal criterion novelty search in a novel sandbox game domain that naturally implements a minimal fitness criterion (i.e. avoiding death), unlike most other domains for testing similar algorithms. More importantly, these experiments demonstrate the viability of the SimSim simulator for future work on quality diversity algorithms. For these experiments, evolution begins from a single seed level that meets some, but not all, Sim needs. Figure 2 depicts this basic room as rendered in The Sims 3. Note that SimSim objects actually only occupy one tile each. The challenge for evolution is to find objects that satisfy the Sim's needs and also don't kill the Sim (by i.e. removing objects necessary for basic survival without replacing them with objects satisfying equivalent needs).

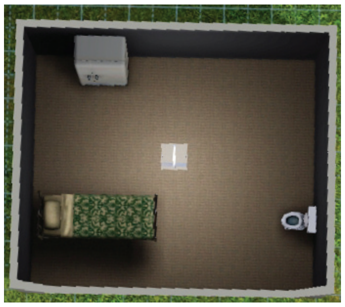


Figure 2: Minimal initial seed room for evolution. If the evolutionary algorithm removes any objects in this room without replacing them with objects satisfying equivalent needs, the Sim cannot survive, thereby earning 0 fitness.

The following algorithms are compared:

- **Novelty search (NS)**, shown in Algorithm 2. The objects in each room are encoded as vectors of objects. The vector distance between rooms in the population and novelty archive is then calculated and used for the k-nearest neighbor calculation in the vector space. A generated house is considered novel if the distance to previously discovered houses exceeds a fixed threshold. This new house is then added to the novelty archive.
- **Minimal Criterion Novelty Search (MCNS)**. In this modification to novelty search, individuals in the population (i.e. rooms) are given a fitness of 0 if their earned fitness is < 0.1. This algorithm is motivated by the binary conception of fitness observed in biological evolution.
- **Minimal Criterion Novelty Search with higher fitness threshold (MCNS-H)**. In this case, fitness is reduced to 0 if an individual earns < 0.2 fitness.
- **(1+1) Evolutionary Algorithm**, shown in Algorithm 3. This algorithm was selected as an example of a canonical greedy evolutionary algorithm, also known as a "hill-climber", that optimizes solutions to a problem using extremely incremental local search, and is a standard comparison algorithm for evolutionary computation research. The main purpose of including it in experiments herein is to see if the domain forces algorithms to sacrifice quality for the sake of diversity. The population consists of one parent and its single child, with the more fit of the two serving as the sole parent of the next two-member generation.

Regardless of the algorithm being tested, the fitness value of a room is based on the total need values divided by the maximum total possible for all the values. The following formula represents the fitness value calculated:

$$f = \frac{b + f + h + s + e + y}{60} \tag{1}$$

where f is the final fitness value and b, f, h, s, e, and y are the final Bladder, Fun, Hunger, Social, Energy, and Hygiene values respectively. If a Sim "dies" before the end of the ticks, the fitness value for the house is set to 0. The maximum fitness value is 1.

Algorithm 2: Novelty Search

```
Input: $popSize, generations$
$curGen$ = 0;
$initHouse$ = House(toilet, bed, fridge);
$pop$ = [] ;                    // house population
$novPop$ = [] ;                  // novelty archive
for $i$ in $populationSize$ do
    $mutHouse$ = mutate($initHouse$);
    add $mutHouse$ to $pop$;
/* Start the evolution                          */
while $curGen < generations$ do
    /* Run game and check novelty              */
    for $h$ in $pop$ do
        $testSim$ = SimAgent();
        $fitness$ = simulate($h$,$testSim$);
        if isNovel($h, fitness, novPop$) then
            add $h$ to $novPop$;
    /* Randomly select houses from
       novel archive and current
       population, mutate the parents,
       and add to the new population
       */
    $newPop$ = [];
    $s$ = $popSize$/6;
    $novParents$ = random($novPop$, $s$);
    for $n$ in $novParent$ do
        for $i$ in 3 do
            $newHouse$ = mutate($n$);
            add $newHouse$ to $newPop$;
    $popParent$ = random($pop$, $s$);
    for $p$ in $popParent$ do
        for $i$ in 3 do
            $newHouse$ = mutate($p$);
            add $newHouse$ to $newPop$;
    /* Replace old population                  */
    $pop$ = $newPop$;
    $curGen$++;
/* Export the novelty archive houses
   */
exportToJSON(novPop);
```

The room genome is a list of furniture objects contained in the room. While not yet explored in this paper, multi-room houses can also be created, and they are encoded as lists of rooms. Room mutations include moving an existing object, deleting an existing object, and adding a new object, with probabilities 0.40, 0.10, and 0.30 respectively. New objects are added at random positions. If there is a collision with an existing object at the selected location, the system will attempt to place the new object in a neighboring location, but will abort mutation if all neighboring tiles are occupied.

Each algorithm was evaluated over 20 runs, each through 100,000 generated individuals. Population size for all NS and MCNS runs was 100, with 1,000 generations. This end time was chosen based on a smaller set of preliminary experiments through 500,000 individuals, which showed that the algorithms tended to converge by 100,000 individuals. (1+1)-EA runs also lasted through 100,000 individuals, but population size was 2. The minimum K-nearest neighbor distance was 3. House size was $7x7$ tiles and Sims lived for 100 ticks.

Algorithm 3: (1+1) Evolutionary Search

```
Input: $generations$
$curGen$ = 0;
/* Create initial house and get the
   fitness                                */
$initHouse$ = House(toilet, bed, fridge);
$pop$ = [] ;                // house population
$startSim$ = SimAgent();
$initFitness$ = simulate($h$,$startSim$);
/* Set the initial best house             */
$bestHouse = initHouse$;
$bestFitness = initFitness$;
/* Start the evolution                    */
while $curGen < generations$ do
    /* Copy and mutate the best house
       */
    $childHouse$ = mutate(House($bestHouse$));
    $childSim$ = SimAgent();
    $childFit$ = simulate($childHouse$,$childSim$);
    /* Replace the best house with the
       better child                       */
    if $childFit > bestFitness$ then
        $bestHouse = childHouse$;
        $bestFitness = childFitness$;
    $curGen$++;
/* Export the best house                  */
exportToJSON(bestHouse);
```

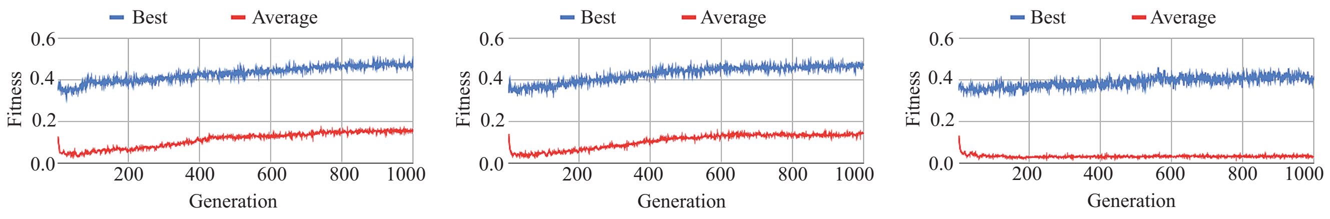
Results

Table 1 shows quantitative results for all runs. A successful algorithm applied to this domain would find a wide variety of houses that simple agents can live in. Figures 3c and 4 provide visual graphs of average house fitness over time and average number of novel houses found.

Figure 3 shows average fitness per generation in each experiment. With a minimum viability criterion implemented, the population's average fitness slowly increased each gen-

Result	NS	MCNS	MCNS-H	(1+1)
Avg archive	181.29	62.8	62	N/A
Std dev	25.94	32.44	25.31	N/A
Avg best fitness	0.56	0.57	0.57	0.63
Std dev	0.033	0.033	0.026	.0163
Best fitness	0.63	0.63	0.63	0.66

Table 1: Experimental results. The novelty search variants find a variety of houses with a slight quality sacrifice.



(a) MC = 0.2 (b) MC = 0.1 (c) MC = 0.0

Figure 3: Average (red) and best (blue) fitness over time. Imposing *no* minimal fitness requirement causes a stark decline in average fitness, highlighting the utility of minimal-criterion-based evolutionary algorithms when searching for collections of artifacts.

eration but began to level off around the 500th generation. At 1000 generations with 100,000 individuals, novelty searches with a minimum criteria value of 0.2 and 0.1 produced a population average fitness of 0.162 and 0.139 respectively. Compared to the novelty search with no minimum criteria value, the population average fitness dropped after the first generation and stayed low. This pure novelty search produced an average population fitness of 0.033.

The blue lines in Figure 3 show the best fitness per generation for each MCNS experiment. The best fitness of each population gradually increased as they evolved and more novel houses were found and remained relatively unchanged for all three minimum criteria values. For the minimum criteria values of 0.2, 0.1, and 0.0 the best fitness from each population per generation was 0.568, 0.574, and 0.564 respectively. Average best fitness at end of run for minimum criteria values of 0.2, 0.1, and 0.0 was 0.568 ($\sigma = .0.026$), 0.574 ($\sigma = .0.033$), and 0.564 ($\sigma = .0033$) respectively. For the (1+1) evolutionary algorithm, the average best fitness found for the final house generated was 0.63 ($\sigma = .0163$) with the maximum best fitness found being 0.667. Figure 6 shows the average fitness per generation.

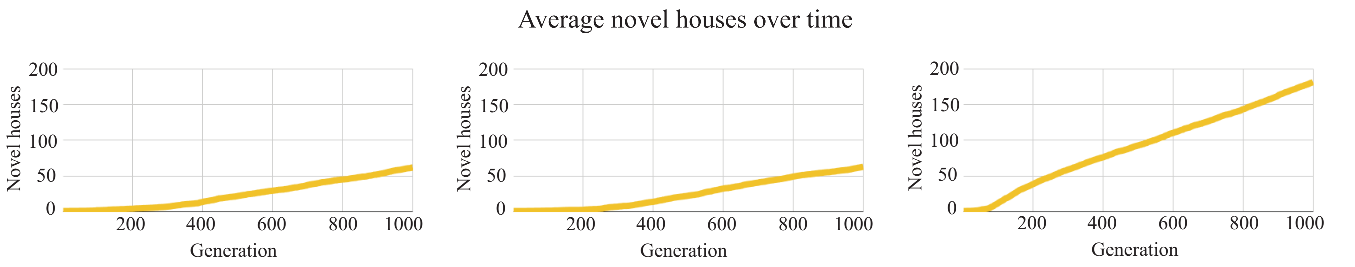
While fitness information informs our understanding of whether or not the implemented algorithms sacrifice quality for the sake of diversity, the intended algorithmic challenge in this domain is for a single run of a search algorithm to find a diverse set of viable designs, not just a single best room. Figure 4 shows the number of novel houses found per generation of the three experiments. In the two experiments with algorithms incorporating a minimal fitness criterion, the population's average fitness slowly increased each generation. While the graphs could potentially imply that fitness is still increasing, a smaller set of preliminary experiments indicated that fitness actually begins to level off around the 500th generation. The average final number of novel houses found by the searches with minimum criteria values of 0.2 and 0.1 were 62 ($\sigma = 25.31$) and 62.8 ($\sigma = 32.44$) respectively, while the final number of novel houses for the search with a minimum criteria value of 0.0 was 181.28 ($\sigma = 25.94$). No number of novel houses is reported for the (1+1) evolutionary algorithm because it does not maintain an archive.

Discussion

The purpose of the experiments in this paper was to investigate minimal criterion search in a domain that has a natural minimal criterion for success (avoiding death), which is a novel challenge tailored for divergent evolutionary search algorithms (which are becoming increasingly popular for PCG). This fitness criterion was then reinterpreted as a tunable parameter for exploring minimal-criterion-based divergent search. In the end, the fitness thresholds chosen did not have significant impacts on best fitness or number of houses found, but having *some* minimal fitness criterion on top of pure novelty search did result in a higher average fitness. It is also interesting to note that the hillclimber tended to find the best fitness, suggesting that the design problem in its current form is not deceptive. However, finding a single house with high fitness, while interesting, frames the goal of evolution as solving a single-objective optimization problem. The fact still stands that a single run of the hillclimber is not nearly as generative as i.e. novelty-based methods, which discover entire archives of viable artifacts. None of the houses generated were able to reach a maximum fitness of 1.0 due to the fact that none of the placeable objects could max out any one need value for the agent. The Sims agent only interacted with an object enough to raise their diminished need above the set threshold value.

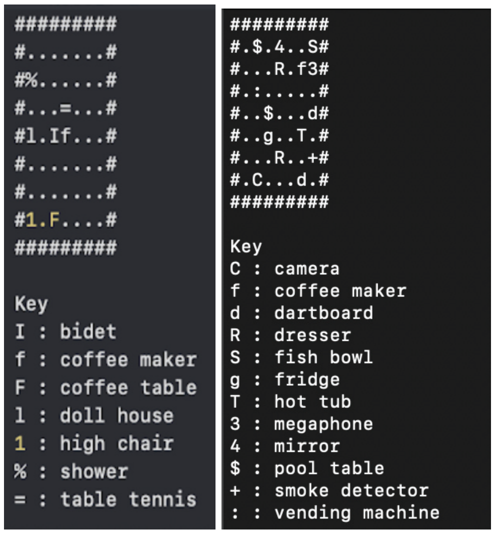
The system tended to replace the three starter objects in the house (a bed, toilet, and fridge) with more need-efficient objects included in the object list. This tendency demonstratseshow the algorithm was able to select which objects were the "best" objects for a house. For example, the "coffee maker" object replaced beds in 50% of the final houses produced because the coffee maker renewed more of the Energy need than the bed object and also replenished Hunger. Similarly, in 25% of the final houses, a "bidet" replaced the starting toilet object. While this object replenished the same amount of Bladder as the toilet, the bidet also replenished Fun and thus proved to be a more beneficial object in the house. Needs that were also not addressed by the starter objects, such as the Fun and Social needs, were also optimized through the hillclimber algorithm. In 70% of the final houses, either a "foosball table" or "table tennis" object was placed because these objects best satisfied both the Social and Fun needs.

Finding the optimal objects in the object lists given to the



(a) MC = 0.2 (b) MC = 0.1 (c) MC = 0.0

Figure 4: Average number of novel houses over time. While pure novelty search finds more novel houses than versions with an imposed minimal criterion (MC), the effect of tested MC values is minimal.



```
#########
#.......#
#%......#
#...=...#
#1.If...#
#.......#
#.......#
#1.F....#
#########


Key
I : bidet
f : coffee maker
F : coffee table
l : doll house
1 : high chair
% : shower
= : table tennis
```

```
#########
#.$.4..S#
#...R.f3#
#.:.....#
#..$...d#
#..g..T.#
#...R..+#
#.C...d.#
#########

Key
C : camera
f : coffee maker
d : dartboard
R : dresser
S : fish bowl
g : fridge
T : hot tub
3 : megaphone
4 : mirror
$ : pool table
+ : smoke detector
: : vending machine
```

Figure 5: Sample houses evolved by the (1+1)-EA and Novelty Search, respectively. Houses are output as JSON that can then be rendered as ASCII. "Optimal" houses found by the greedy (1+1)-EA prefer a few specific maximally need-satisfying items, while "novel" houses include a wider variety of objects that still satisfy the Sim's needs.

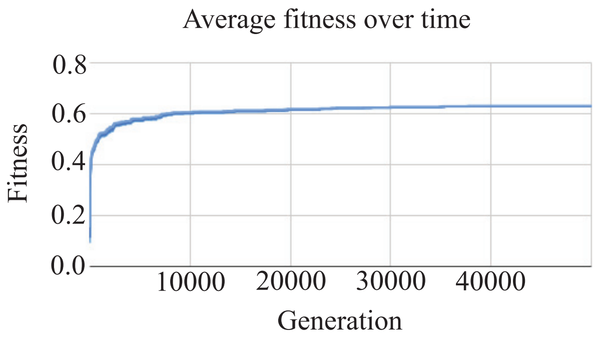


Figure 6: (1+1) EA average fitness per generation. This algorithm converges early in its run and only shows incremental improvement given more time. Note that the EA returns just *one* interesting room instead of many.

mutators could help to identify any imbalances that might occur from including or excluding certain objects in the house. The case study presented in this paper adds to the ever-growing collection of domains where evolution, and specifically evolution-based procedural content generation, proves useful for automated analysis of game design. The notable recurrence of a few very specific furniture items in most high-performing houses would suggest, for instance, nerfing the coffee maker. Algorithms that are explicitly designed to find a diversity of high-performing artifacts are uniquely suited for this role.

In future work, a monetary constraint (like those used in the original Sims games when constructing houses) could be implemented to challenge the algorithm to evolve a house towards optimization given constraints on the object selection. This resource-constrained version of the design problem would provide an interesting challenge for quality diversity algorithms, and may in fact add deceptive elements to the domain. We would also like to experiment with the Sim agent behavior to see how overall house generation is affected. Such experiments could include changing the ranking of needs, allowing diagonal movement, replenishing a need to the maximum value, etc. In addition to adding evolution constraints and altering agent behavior, many other evolution-centric parameters (i.e. mutation probabilities, minimum k-nearest neighbor distance) could be played with to see how the ending novelty archive is affected.

Conclusion

This paper introduced a novel testbed for divergent search search and PCG based on the sandbox game The Sims. Various evolutionary algorithms were implemented as a means to explore house creation space and provide an initial foundation for future work in the sandbox game domain. These algorithms were able to generate hundreds of "Sims-livable" houses with a variety of furnished objects and additionally exposed potential imbalances in the game's design space.

Acknowledgements

This work was supported by the National Science Foundation (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

References

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

Bourse, Y. 2012. Artificial intelligence in the sims series.

Brant, J. C., and Stanley, K. O. 2017. Minimal criterion coevolution: a new approach to open-ended search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 67–74.

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

Fontaine, M. C.; Lee, S.; Soros, L.; De Mesentier Silva, F.; Togelius, J.; and Hoover, A. K. 2019. Mapping hearthstone deck spaces through map-elites with sliding boundaries. In *Proceedings of The Genetic and Evolutionary Computation Conference*, 161–169.

Gomes, J.; Urbano, P.; and Christensen, A. L. 2012. Progressive minimal criteria novelty search. In *Ibero-American Conference on Artificial Intelligence*, 281–290. Springer.

Gravina, D.; Khalifa, A.; Liapis, A.; Togelius, J.; and Yannakakis, G. N. 2019. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Khalifa, A.; Lee, S.; Nealen, A.; and Togelius, J. 2018. Talakat: Bullet hell generation through constrained map-elites. *Proceedings of the Genetic and Evolutionary Computation Conference*.

Liapis, A.; Martınez, H. P.; Togelius, J.; and Yannakakis, G. N. 2013. Transforming exploratory creativity with delenox. In *Proceedings of the Fourth International Conference on Computational Creativity*, 56–63. AAAI Press.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2015. Constrained novelty search: A study on game content generation. *Evolutionary computation* 23(1):101–129.

Salge, C.; Green, M. C.; Canaan, R.; Skwarski, F.; Fritsch, R.; Brightmoore, A.; Ye, S.; Cao, C.; and Togelius, J. 2020. The ai settlement generation challenge in minecraft. *KI-Künstliche Intelligenz* 34(1):19–31.

Wang, R.; Lehman, J.; Clune, J.; and Stanley, K. O. 2019. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 142–151.

Yu, L. F.; Yeung, S. K.; Tang, C. K.; Terzopoulos, D.; Chan, T. F.; and Osher, S. J. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30,(4), July 2011, article no. 86* 30(4).