Mugla Journal of Science and Technology

IMPLEMENTATION OF A GENERIC FRAMEWORK ON CROWD SIMULATION: A NEW ENVIRONMENT TO MODEL CROWD BEHAVIOR AND DESIGN VIDEO GAMES

Furkan Yücel, Department of Modeling and Simulation, Graduate School of Informatics, Middle East Technical University, Turkey, furkan.yucel@metu.edu.tr
(https://orcid.org/0000-0001-7522-6248)
Elif Sürer*, Department of Modeling and Simulation, Graduate School of Informatics, Middle East Technical University, Turkey, elifs@metu.edu.tr
(https://orcid.org/0000-0002-0738-6669)

Received: 21.07.2020, Accepted: 09.10.2020 | Research Article
*Corresponding author | DOI: 10.22531/muglajsci.706841

Abstract

Crowd behavior is the collective act and gathering of a group of individuals to achieve a shared purpose. Swarm intelligence-based optimization algorithms are usually used to solve complex problems for crowd behavior. Crowd simulations are often used for the analyses that require precision in different domains such as complex structural analysis, image recognition, creating nature-inspired non-player character movements in video games, and more. In this study, a generic crowd simulation framework that can be used to simulate already-available crowd simulation algorithms and design new ones was developed. The test environment layout was generated with the use of a generate-and-test algorithm combined with the crowd simulation algorithms to make sure that the generated content is meeting the requirements of a crowd simulation environment. Within the framework, three different crowd simulation algorithms —firefly algorithm, particle swarm optimization, and artificial bee colony— are generated and also implemented as puzzle-like video games. The results show that all fireflies achieved to gather at the global minimum of the generated layout faster and in a more precise way than the artificial bee colony algorithm and particle swarm optimization algorithm. The developed framework enables a generic and parametric testbed to design and compare different algorithms and to generate video games.

Keywords: Crowd Simulation, Firefly Algorithm, Artificial Bee Colony Algorithm, Particle Swarm Optimization, Generate-and-Test Algorithm, Game Generation

KALABALIK SİMÜLASYONU ÜZERİNE GENEL BİR ÇERÇEVE GELİŞTİRİLMESİ: KALABALIK DAVRANIŞINI MODELLEMEK VE VİDEO OYUNLARI TASARLAMAK İÇİN YENİ BİR ORTAM

Özet

Kalabalık davranışı, ortak bir amaca ulaşmak için bir grup bireyin kolektif eylemi ve toplanması olarak tanımlanmaktadır. Sürü zekası tabanlı optimizasyon algoritmaları genellikle kalabalık davranışı ile ilgili karmaşık problemleri çözmek için kullanılmaktadır. Kalabalık simülasyonları genellikle hassas analiz gerektiren karmaşık yapısal analiz, görüntü tanıma, video oyunlarında doğadan ilham alan oyuncu olmayan karakter hareketleri oluşturma gibi farklı alanlarda kullanılmaktadır. Bu çalışmada, halihazırda mevcut olan kalabalık simülasyon algoritmalarını simüle etmek ve yenilerini tasarlamak için kullanılabilecek genel bir kalabalık simülasyon çerçevesi geliştirilmiştir. Test ortamı düzeni, oluşturulan içeriğin bir kalabalık simülasyon ortamının gereksinimlerini karşıladığından emin olmak için kalabalık simülasyon algoritmaları ile birleştirilmiş bir oluştur-ve-test-et algoritması kullanılarak oluşturulmuştur. Bu çerçevede, üç farklı kalabalık simülasyon algoritması (ateşböceği algoritması, parçacık sürü optimizasyonu ve yapay arı kolonisi) üretilmekte ve bulmaca benzeri video oyunları olarak da geliştirilmektedir. Sonuçlar, tüm ateşböceklerinin üretilen düzenin global minimumunda toplanmayı, yapay arı kolonisi algoritması ve parçacık sürü optimizasyonu algoritmasından daha hızlı ve daha kesin bir şekilde başardığını göstermektedir. Geliştirilen çerçeve, genel ve parametrik bir test ortamının farklı algoritmalar tasarlanıp karşılaştırılması ve video oyunları geliştirilme amacıyla kullanılabilmektedir.

Anahtar Kelimeler: Kalabalık Simülasyonu, Ateşböceği Algoritması, Yapay Arı Kolonisi, Parçacık Sürü Optimizasyonu, Oluştur-ve-Test-Et Algoritması, Oyun Üretimi

Cite

Yücel, F. and Sürer, E. (2020). "Implementation of a Generic Framework on Crowd Simulation: A New Environment to Model Crowd Behavior and Design Video Games", Mugla Journal of Science and Technology, 6(2), 69-78.

1. Introduction

Crowd simulations are mainly used in order to create realistic mass behaviors in many fields such as computer games and movies (i.e., visual effects or animations) [1], to predict and simulate crowd movement in emergency situations (i.e., fire escape training and earthquake evacuation) [2], and to simulate military strategies in specific scenarios [3]. There are different algorithms and studies that simulate crowd behavior in the literature, nature-inspired or bio-inspired optimization algorithms being state-of-the-art. These swarm behavior-based optimization algorithms such as artificial bee colony, particle swarm optimization, and ant colony optimization were widely used in the literature given their applicability to different research domains. Yang [4] examined the use of the firefly algorithm (FA) in the field of optimization and engineering. The purpose of this examination was to check the effectiveness of FA against some traditional algorithms. As a result, FA showed better performance over traditional algorithms in terms of performance and finding global and local maxima and minima. In a book edited by Dey [5], many applications of FA were held in different research areas such as digital image processing, electrochemical-based machine processes, structural damage identification, and face recognition. In another recent study [6], the authors developed an evacuation simulation based on a bat algorithm, which is an optimization algorithm created from the inspiration taken from the bats in real-life. To compare the results of the proposed algorithm, a particle swarm optimization-based evacuation simulation was also developed. The results showed that the bat algorithm outperformed the particle swarm optimization in terms of speed. In another study on bio-inspired optimization by Wang et al. [7], a monarch butterfly optimization was introduced. The algorithm was inspired by real-life monarch butterflies that travel from northern parts of the USA to Mexico. While introducing the algorithm, two lands that represent the USA and Mexico are used in order to simulate the movement of the butterflies. Genetic algorithms were used to create new and fitter generations. While creating those new generations, the authors wanted to keep the total population size the same, so they replaced old monarch butterflies with the new ones if the new one was fitter than the old one. In another bio-inspired optimization study, Darwish [8] had done analyses on nine different bio-inspired algorithms and discussed their field of work, how they work, and what strategy makes them efficient. Darwish also mentioned the similarities between these algorithms, and he suggested to hybridize them with each other and with other methods.

The use of bio-inspired optimization algorithms is also common in video games. In a recent study done by Kowalski et al. [9], the authors developed a video game, specifically a car race genre, where they used a swarm-based meta-heuristic called “krill herd algorithm” to create movement strategies for the bots in the game. Krill herd algorithm uses genetic operators to optimize the goal if the task is minimization or maximization. In another work by Diaz et al. [10], the authors implemented particle swarm optimization to optimize and determine non-player characters’ (NPCs) movements in a first-person shooter genre. Ponticorvo et al. [11] studied bio-inspired computational models for game mechanics, where the user interacts with the game by use of those models in the setting of educational games and serious games. As the studies show, the use of bio-inspired optimization algorithms on modeling artificial intelligence in video games is a promising field, both in terms of game generation and NPC creation.

Although bio-inspired optimization algorithms are highly varied and used in the video game research, an end-to-end simulation and game generation framework is missing. The abovementioned studies and developed frameworks either work as simulators that implement already-available algorithms or as video games where the bio-inspired optimization algorithms are already embedded into the gameplay. The simulators and their comparative analyses mainly focus on comparing the already-available algorithms in different domains without the flexibility of modifying or enhancing those algorithms. The algorithm generation and game generation tasks are performed in separate environments, and a dynamic interface that feeds the algorithms to the games is not available. Thus, the literature lacks an end-to-end generic framework that gives the users the autonomy of generating and testing new optimization algorithms followed by an automated video game generator that produces, measures, and visualizes the flocking behavior while testing the interaction behaviors of NPCs and human players. Such a framework would also mold the roles and professions of game developers, algorithm generators, and game designers into one, with an easy-to-use and modular interface allowing experiments in the three phases of the development cycle —algorithm generation, simulation, and game generation.

In this study, to overcome the abovementioned limitations, we developed a generic framework in an attempt to understand and exploit the common characteristics of bio-inspired optimization algorithms to provide a fast and useful tool, which creates crowd simulations by using pre-defined parameters without coding the algorithm itself, which allows the users to combine the available parameters while also customizing the code with their scripts. This generic framework on crowd simulation was developed to speed up the process of creating and implementing bio-inspired optimization algorithms to allow the developers and designers to experiment with the parameters during algorithm implementation, followed by developing games using the implementation results. To test and validate the framework’s performance in these aspects, three state-of-the-art crowd simulation algorithm-based video games, where the objective is mainly on creating a puzzle and simulating the behavior of the NPC movements by crowd simulation, were developed. Also, to show the flexibility of the framework, artificial attractors were implemented in the games, which were designed to modify both the location of the global minima and the movement of particles in the algorithms and the games. The main tasks are modifying the parameters to change the location of the global minimum of any optimization function that would be selected to create a game environment and successfully simulating the crowd behavior of the particles, which should collectively gather at a global minimum, a global maximum or the most attractive daisy for any generated layout. The results show that the comparative analysis of the bio-inspired optimization algorithms generated with our framework showed similar results to the studies in the literature [12], while also easily generating video games using the parameters of the algorithm generation and simulation phases. Thus, this framework can provide an end-to-end algorithm generation, algorithm testing, and game generation environment for both game developers and game designers, enabling a variety of experiments on the flocking behaviors of NPCs and human players.

2. Materials and Methods

2.1. Crowd Simulation Framework

The crowd simulation framework is developed to create an environment where the developers can implement already existing algorithms from the literature or design new algorithms by selecting different parameters and simulating the outcomes of the selected parameters. To develop such a framework, some common characteristics between bio-inspired algorithms were extracted from the literature. Darwish et al. [8] studied and reviewed many different bio-inspired swarm optimization algorithms, and there are three different main characteristics between those algorithms, such as velocity-based strategies, cost-based strategies, and randomized strategies (Figure 1). The framework is developed with a modular structure that allows users to select and combine different strategies, and that can be improved by the developers according to the need of the game that would be generated.

In order to use the framework first, the developer should specify how many different movement strategies the system has on each iteration. Then, the developer should select the necessary parameters for each movement strategy. These parameters are classified into three groups, such as update strategy, which controls when the particles update themselves; particle class strategy, which controls according to what and how the particles calculate their new positions; and next position strategy, which controls how the particles update their positions throughout the simulation.

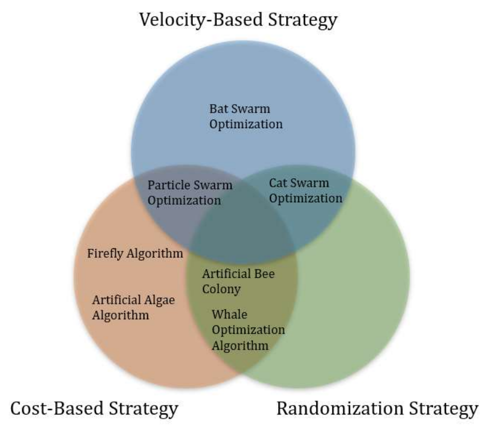

Figure 1. Classification of some of the bio-inspired swarm optimization algorithms.

The update strategy has three Boolean operators —the first one is called when the update is only for a better cost, which allows the particle updates on its position only if the new position is better than the current one. The second one is called "Save global best," which allows the particle to record the global best position and cost. The third parameter, Only update if not updated before, allows the particle update during the iteration only if the previous try fails to find a better position for the particle. If no Boolean operator is selected, the particle updates itself on every iteration.

The particle strategy also has different Boolean operators. The first one, Check against no particle, allows the particles to be independent of others. The second parameter, Check against better particles, allows the particles to use other particles that have better cost outcomes than theirs; when a new position is being calculated. The third parameter, Completely random particle, allows the particle to select a random particle and calculate its new position accordingly.

The next position calculation strategy has different parameters. The first one, Is Exponential, controls the exponentiality while calculating a new position. The second one, Should use distance, checks if the calculations include distance as a parameter, and it also has two different numeric parameters connected to itself, which are the power of distance and distance multiplier. Both parameters control the effect of the distance to the calculation. The third parameter, Should use vectors, allows the particle to use velocity while calculating a new position, and it is connected to a numeric parameter called the constant multiplier, which controls the effect of the velocity to the calculation. The damping strategy controls the maximum magnitude of the velocity. The parameter, Should check the global best parameter, allows the particle to use the global best position while calculating a new position, and should check the personal best that allows the particle to use its best location. Randomness property allows the particle to use a random multiplier while calculating a new position, and random vector properties allow the particle to randomly search for a new location.

After all the parameters are set, the simulation starts and continues in an order, which is first, the particles are generated, then the initial cost values are calculated. After these two steps, iterations start, and update and movement strategies are executed in that order until the maximum iteration limit has been reached (Figure 2). The framework is generic enough to create and test different crowd simulation algorithms, and, in this study, three different crowd simulation algorithms, firefly algorithm, particle swarm optimization, and artificial bee colony, were implemented as use cases. Also, the framework allows users to create puzzle video games out of each created algorithm.

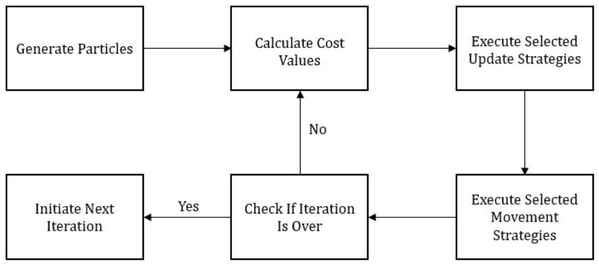
Generate Particles
Calculate Cost Values
Execute Selected Update Strategies
No
Initiate Next Iteration
Yes
Check If Iteration Is Over
Execute Selected Movement Strategies

Figure 2. The workflow diagram of the algorithm implementation.

2.2. Simulation of Fireflies

Firefly algorithm [4] (FA) is a metaheuristic optimization algorithm that has three main rules, which are described as follows: First of all, all fireflies in the simulation are unisex, and they can be attracted to any other firefly. Secondly, fireflies are attracted to other fireflies which are brighter than themselves; attractiveness β is proportional to the Cartesian distance r, which is the Cartesian distance between the two fireflies and to β_0 which is the base attraction coefficient and to γ which is the light absorption coefficient:

$$\beta = \beta_0 e^{-\gamma r^2} \qquad (1)$$

Lastly, brightness I is proportional to the Cartesian distance r of the firefly to the global minimum of the objective function, I_0, which is the base light intensity coefficient and the light absorption coefficient γ.

$$I = I_0 e^{-\gamma r^2} \qquad (2)$$

The movement of the fireflies depends on the Cartesian distance between the two fireflies and their light intensities, and randomization is also used with a vector that is created by Gaussian distribution. The algorithm iterates until the threshold of the maximum number of generations is reached. An example of a running firefly algorithm that has not achieved the maximum number of generations yet can be seen in Figure 3. There has been a pace coefficient θ implemented by the authors, which slows down the fireflies in the framework and in the game in order to make the firefly movement visible to the player. During the tests, the pacing coefficient θ was set to one, but during the game, the pacing coefficient θ was set to twice the total number of fireflies in order to enhance the game difficulty.

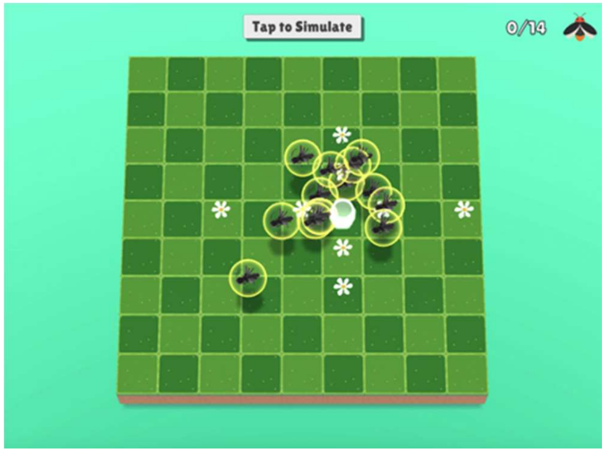

Figure 3. The behavior of the fireflies after ten iterations —they are almost gathered at the location of the jar.

The pace coefficient θ is directly proportional to the number of fireflies. This is a result of the fact that the firefly which has the least amount of light intensity tries to move towards the other fireflies, besides itself, since the other fireflies are brighter and closer to the global minimum of the optimization function, and the fireflies are attracted to the light intensity. Therefore, it moves with a greater distance at each iteration if the system has more fireflies in it. The pace coefficient θ is for visualization purposes only. The new location of a particle is calculated as follows:

$$x_i = x_i + \frac{(\beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\varepsilon_i)}{\theta} \qquad (3)$$

Randomization is a crucial part of the crowd simulation in order for the particles to search for a minimum or maximum. The randomness coefficient α, which has a value of 0.05, is used for randomization while a vector uniformly created by the Gaussian distribution is used in order to calculate the new locations of the fireflies.

2.3. Daisy Attraction

Daisy attraction d_i is calculated differently than the firefly attraction. The reason for this situation is that firefly attraction comes from the light intensity while the daisy attraction comes from the surrounding daisy population, so the firefly attraction mostly depends on the Cartesian distance to the global minimum, whereas daisy attraction is calculated with the Cartesian distance between the daisies,

$$d_i = \sum_{k=0}^{n} d_0 \, e^{-\gamma(x_i - x_k)^2} \qquad (4)$$

where d_0 is the base daisy attraction. Firefly algorithm with daisy attraction can be explained with a pseudo-code as follows:

Firefly algorithm with the daisy attraction

```
Objective function $f$(x), x = $(x_1, \dots, x_d)$
1. Initialize a population of fireflies $x_i (i = 1,2, \dots, n)$
2. Initialize a population of daisies $y_i (i = 1, 2, \dots, k)$
3. Simulate all fireflies
while (t<iteration limit)
for i=1: n all n fireflies
  for j = 1: n+k all n fireflies and k daisies
    Light intensity $I_i$ at $x_i$ is determined by $f(x_i)$
    if ($I_j$>$I_i$)
    Move the firefly I towards j in all d dimensions
    end if
    if ($d_j > I_i$)
    Move the firefly I towards daisy (j-n) in all d
    dimensions
    end if
    Update the light intensity
  end for j
end for i
end while
4. Determine win or lose
```

2.4. Relocation of the Global Minimum

To create gameplay that is suitable for a crowd simulation-based game, the term relocation of a global minimum was introduced. The global minimum would be the offset at the start of the simulation with the offset vector $\vec{O}$. The offset vector is calculated as

$$\vec{O} = \frac{\sum_{i=0}^{k} \vec{d_i}}{k} \tag{5}$$

where k is the total number of daisies and $\vec{d_i}$ is the location vector of the daisies.

Ackley function [13] was created to generate a hill-like mathematical geometry to test the learning strategies. The function has one global minimum at (0,0,0) and is symmetrical on both x and y axis. The objective function is selected to be the Ackley function due to the fact that the function is open to simple adjustments without any direct changes to the function itself —such as changing the global minimum's location or scaling the function, which would change the attraction values of fireflies (Figure 4). The offset vector, which is added to the formula directly as an addition to the location of the firefly in the system, can be seen in the formula below:

$$f(x) = -20e^{\left[-\frac{1}{5}\sqrt{\frac{1}{k}\sum_{i=1}^{k}(x_i - o_i)^2}\right]} \tag{6}$$

where k is the number of dimensions. Ackley function with the offset vector being zero can be seen in Figure 5.

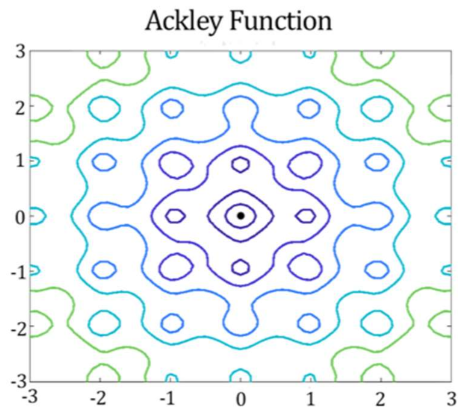

Figure 4. The graphical representation of the Ackley function where the black dot in the middle represents the global minimum of the function.

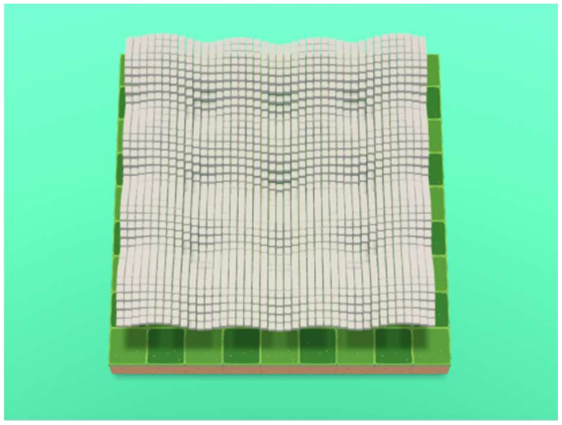

Figure 5. The outline of the Ackley function with the offset vector being (0,0,0).

2.5. Generate-and-Test Method

Since the game levels are created procedurally, and the layout is not affecting the difficulty of the game level, the generate-and-test method is selected as the generation algorithm. Togelius et al. [14] introduce the generate-and-test algorithm as an algorithm consisting of both a generation and a test structure. Following the content generation, it is tested with some criteria to check if the generated content satisfies the requirements of design — i.e., if there are overlapping objects, or if the level is playable. If the generated content fails the test, it is recreated until it passes the test. In our game, we used the generate-and-test algorithm for each generated content, which is explained in the next section.

2.6. Level Generation

The firefly video game requires three components: the fireflies, a jar as a target for the fireflies, and daisies to act as artificial attractors. Fireflies are generated randomly without overlapping with each other, which is controlled by the generate-and-test algorithm, in the given area limits that would fit in the game environment, which is from -2 to +2 for all three dimensions. The light intensity of each firefly is calculated with the objective function, which is, in our case, the Ackley function. The grid of the game is selected to be located between -3 to 3 both in x and z directions and at 0 in y-direction since the global minimum is located at the 0 in y-direction at Unity 3D game engine [15]. Daisies are generated by the use of a generate-and-test algorithm where the algorithm creates a layout by generating daisies on random grid locations and then tests the generated layout with the given rules. These given rules are, in our case, do daisies overlap with each other or with the fireflies and how they affect the global minimum of the objective function. The algorithm stops iterating if the test result is sufficient enough to generate a layout for the game itself, and if not, this process is repeated until the given rules are satisfied. Lastly, a jar is created on a tile which is not at the global minimum, and which does not have a daisy on it to avoid creating a puzzle level which is already solved. Figure 6 shows an example of a generated game level.

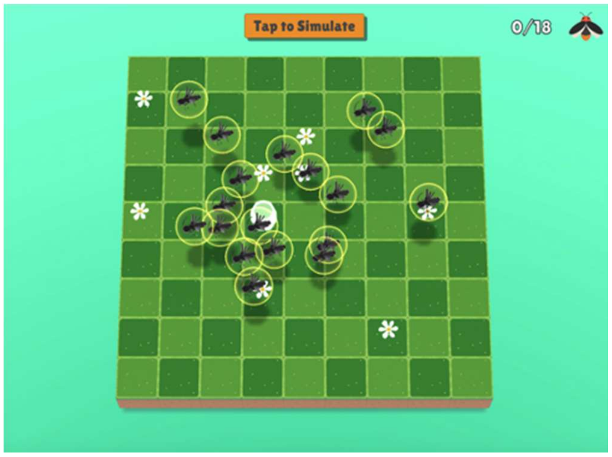

Figure 6. An example of a generated level outline with fireflies, daisies, and a jar in the game.

The game consists of many different levels, which are created procedurally at the start of each session. There are two parameters that are modified at the start of each level —the required number of fireflies that have to be created and collected. Firefly count for each level is selected randomly between the numbers of 10 and 20. The number of daisies was selected to be an even number between the numbers of 4 and 8 in order to create a level balance. If the daisy number is more than 8, the difficulty increases since daisies become dense, and one of the goals of the game is keeping the daisies separated. The other goal is to create symmetry with the daisies, so an odd number of daisies create a problem where players cannot create direct symmetrical layouts, so the game level difficulty increases.

2.7. Gameplay

The game is developed as a puzzle game based on the firefly algorithm for game rules and simulations of the results. The player has the goal of catching the firefly into the jar by rearranging the daisies on the grid layout. The game is designed to be played on mobile platforms, so the gameplay consists of tap, drag, and drop actions. The game world consists of 9-by-9 tiles, daisies, and a jar, which are all located at the center of the tiles. Tiles can contain at most one daisy. The player interacts with the game as he/she reorganizes the arrangement of the daisies by simply dragging and dropping daisies to empty tiles. If the selected daisy is dropped on a tile, which has a daisy on it, that daisy returns to its initial position because only one daisy can be present on a single tile. To start the simulation, the player must press on the button, “tap to simulate”. After the tap occurs, the firefly algorithm starts to iterate until the fireflies gather at a location, which can be the jar, the global minimum, which is not the jar or the daisy with the highest attraction value.

The game consists of puzzles that are generated at the start of each level, and the puzzle is created with two different effects of daisies, which are the relocations of the global minimum due to the average position of daisies and the daisy attraction. The player rearranges the daisies to solve the puzzle by drag and drop action. The player can navigate through the game menu with an initial screen having one button that leads the player to the game. Throughout the game, the player can check the total amount of fireflies and the number of the collected fireflies from the upper right part of the screen. If the player wins, a similar panel appears, which leads the player to the next level or to the main menu. If the player fails, the “game over” panel appears to lead the player either to restart the game or to the main menu.

2.8. Game Rules

The main aim of the game is to collect the fireflies in the jar, as shown in Figure 7. After the player arranges the daisies and taps on the “tap to simulate” button, the firefly algorithm starts to iterate with a delay of 0.5 seconds due to the fact that at the iteration zero, the fireflies have to be relocated with respect to the newly calculated global minimum since the daisies are rearranged by the player before the iterations start. In order to relocate the fireflies with animation instead of teleporting them to new locations, the LeanTween [16] plugin from the Unity Asset Store is used. If there are three or more daisies on adjacent tiles, the fireflies will gather at the location of those daisies due to daisy attraction, and then the player fails. If not, fireflies gather at the global minimum, and if the arrangement is correct and the recalculated position of the global minimum is equal to the jar’s location, the player wins as demonstrated in Figure 7. Otherwise, after 175 iterations, the player loses, as can be seen in Figure 8.

The performance tests were run on a MacBook Pro 12.1, with 8 Gb RAM, with Intel Core i5 processor 2.7 Ghz, and for the algorithm tests, MATLAB_R2019b was used. Unity version 2019.2.0f1 was used for the development of the game, and some of the assets such as 3D models, sprites, textures were modeled or drawn by the authors while the assets were taken from [17].

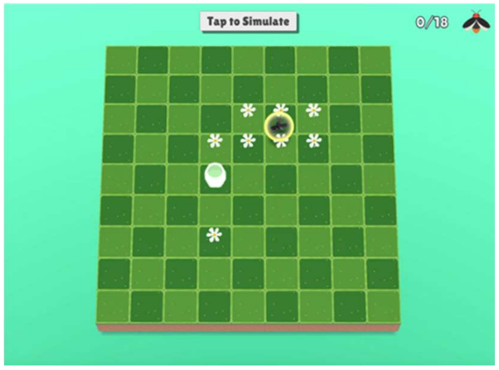

Figure 7. When the fireflies gather at the jar, the player wins.

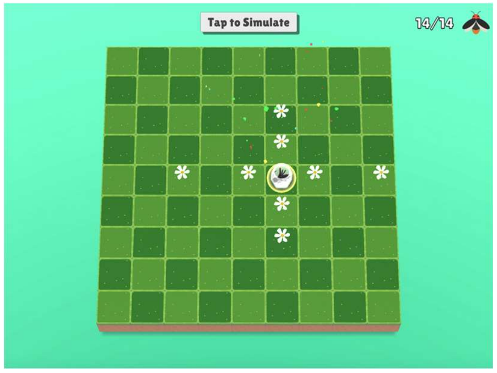

Figure 8. When the fireflies gather at the most attractive daisy, the player loses.

3. Results

With the proposed framework, we created three puzzle games using a firefly algorithm, particle swarm optimization, and an artificial bee colony for the simulation results of the puzzle, as can be seen in Figure 4, Figure 9, and Figure 10.

In order to check the reliability of our framework, we tested the Firefly algorithm (FA) against two different algorithms which are Artificial Bee Colony (ABC) and Particle Swarm Optimization (PSO), with different numbers of iterations and with different numbers of particles within the framework as Agarwal et al. [12] compared those in their work. The results are shown below in Table 1, Table 2, and Table 3, respectively. Table 1 shows the effect of the change in the number of particles in the system, whereas Table 2 shows the effect of the change in the total number of iterations. Figure 11, Figure 12, and Figure 13 show the results of the tests for FA, PSO, and ABC, respectively. The number of particles in the system has a positive effect on locating the global minimum for the FA as well as the ABC but not PSO. Even if the number of particles in the system is increased, the computation time does not increase with a similar ratio.

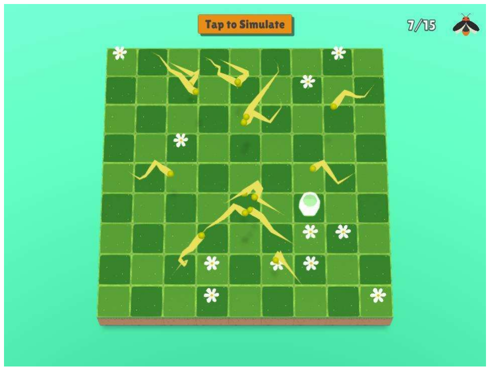

Figure 9. Particle swarm version of the game.

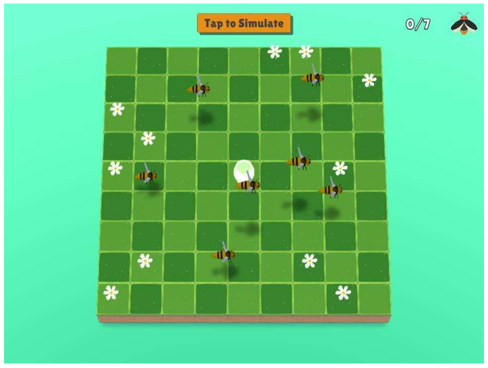

Figure 10. Artificial bee colony version of the game.

The total number of iterations has an impact on all three algorithms; also, as a result of the increase in the iteration amount, the computation time increases with a similar fashion to the increase in the iteration amount. The average cost of each particle in the system decreases at each of the iterations for the FA, whereas it decreases and increases time to time for the PSO and ABC, as can be seen in Figure 14 and Figure 16. Lastly, we tested these three algorithms with a larger population size than the previous test, which is 50 particles in the system, and more iterations, which are 100 iterations at a larger area which is 40-by-40, and the results show that all three algorithms have similar results in terms of cost of the best particle in the system and both the ABC and FA

Table 1. Test results of the Ackley function with different population sizes.

	FA	PSO	ABC	FA	PSO	ABC
Best average cost	0.30	1.60	3.12	0.19	1.56	0.26
Final average cost	0.30	2.50	3.12	0.19	1.69	2.85
Time spent	3.73s	3.41s	6.20s	4.33s	3.83s	7.30s
Population	5	5	5	15	15	15
Iteration	20	20	20	20	20	20

scored better than the PSO in terms of best average cost. However, in terms of computation time, ABC had the highest computation time.

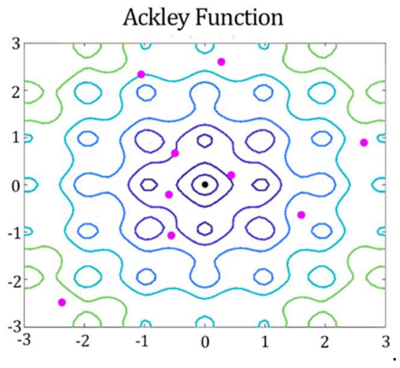

Figure 11. A closer look at an iteration of the firefly algorithm and Ackley function's graph.

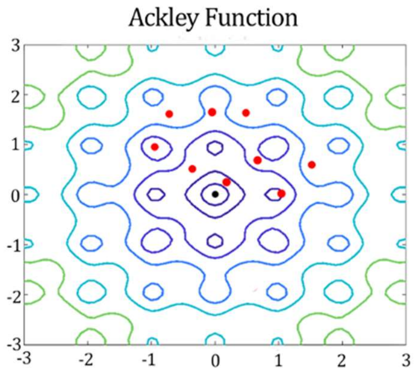

Figure 12. A closer look at an iteration of the particle swarm optimization.

4. Discussion

Artificial bee colony (ABC) works more with a focus on the path-finding. The algorithm was created based on the real-life behaviors of bees. At first, a random food source is created in random locations, and then the bees are created in random locations. The algorithm has three different bee roles, which are employed bee, onlooker bee, and scout bee. Karaboga [18] explains these roles as onlooker bee is a bee that waits to make a decision to choose a food source, and an employed bee is a bee that goes to the food source to a previously visited food source, and a scout bee is a bee that searches for a new food source in a random location.

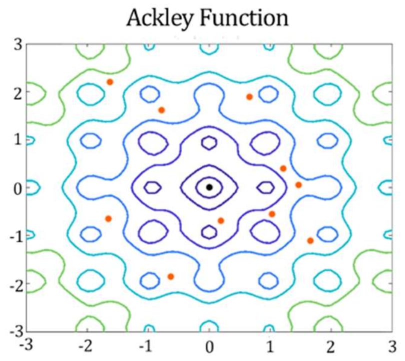

Figure 13. A closer look at an iteration of the firefly algorithm and Ackley function's graph.

The main reason why FA is better at finding the global minimum than ABC is that while ABC iterates only some part of the population while searching for better locations, others either wait or go to the already searched areas. However, in FA, all the fireflies try to reach the maximum light intensity, thus the global minimum. Due to the fact that ABC calculates all bees as if they were employed bees and the algorithm keeps the ones which are improved as employed bees. Then, the rest is calculated as if they were onlookers, and the ones that are not improved in the first two parts are calculated as scouts at each iteration; it takes almost twice the computation time of FA, as shown in Table 1.

FA outranks both PSO and ABC in terms of finding the global minimum with a low population size, which is, in this case, 5 (Table 1). The main reason is that the particles in PSO would consider their next location based on their previous velocity and the best location achieved within the system. Due to the fact that the particles only consider the best location and not all the locations of the particles in the system, the particles have a lower chance of locating the global minimum with a lower number of populations. For ABC, the main problem with the low number of bees in the system is that bees have to communicate and collaborate within the system, but the number is not sufficient enough to operate as a hive, so it fails to locate it. The success of FA comes from the extensive search for brighter light, and this causes the particles to get closer to the global minimum in each term as the fireflies communicate through attractiveness.

In Figure 14, Figure 15, and Figure 16, it can be seen that the particles of FA continuously get closer and closer to the global minimum of the system, whereas both PSO and

Table 2. Test results of the Ackley function with the different number of iterations.

	FA	PSO	ABC	FA	PSO	ABC
Best average cost	0.19	1.56	0.26	0.006	0.36	0.032
Final average cost	0.19	1.69	2.85	0.006	0.49	2.30
Time spent	4.33s	3.83s	7.30s	8.34s	8.03s	15.28s
Population	15	15	15	15	15	15
Iteration	20	20	20	45	45	45

ABC have difficulties collectively getting closer to the global minimum at each iteration during the simulation.

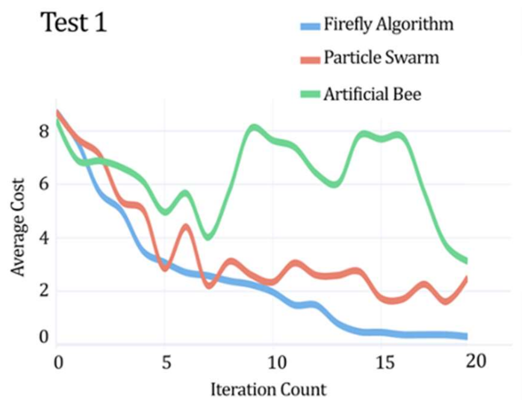

Figure 14. The average cost graph when there are five particles in the system and 20 iterations.

This problem stems from PSO's way of searching for the global minimum, as mentioned before. Since the updated velocity calculation includes both the previous velocity of the particle and the particle's personal best location, it disturbs the particle's focus from the best location in the system. The reason is quite different for the ABC since scout bees are searching for random locations in the system to find new food sources and their cost changes in an irregular way.

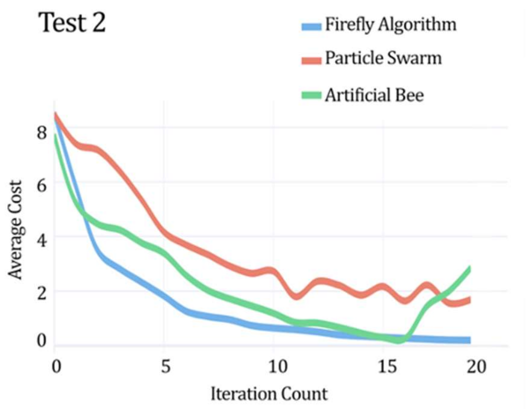

Figure 15. The average cost graph when there are 15 particles in the system and 20 iterations.

By analyzing Table 1, Table 2, Figure 14, Figure 15, and Figure 16, it can be said that both PSO and ABC algorithms fail to preserve the best average location in search of the global minimum, whereas FA succeeds in saving the best position achieved due to the same reasons of fireflies' managing to get closer to the global minimum at each iteration.

Table 3 shows that at a larger field, with more particles and iterations, ABC becomes as accurate as FA in terms of locating the global minimum. The main reason behind the improvement of ABC is that the bees share information with one and another, so with more iterations, they gather more data, and with more population, each bee accesses more data. Another reason why FA is worse than ABC at this test is that, while the fireflies find their own way by attraction, the distance between the fireflies has a direct effect on it.

Table 3. Test results of the Ackley function with 50 particles and 100 iterations at a 40-by-40 field.

	FA	PSO	ABC
Best cost	0.001	0.001	0.001
Best average cost	0.001	0.010	0.001
Time spent	56.51s	50.29s	74.26s

When the fireflies gather at a close position to the global minimum, fireflies get slower than usual. At this test, PSO manages to have the best particle's cost the same as both FA's best particle's cost and ABC's best particle's cost due to the same reason. Even though the best cost of the PSO is better than FA, it still stays behind of FA in terms of the best average cost. When the test results are compared to the results from the work of Agarwal et al. [12], it can be seen that the trend of the algorithms is the same, and the results are similar.

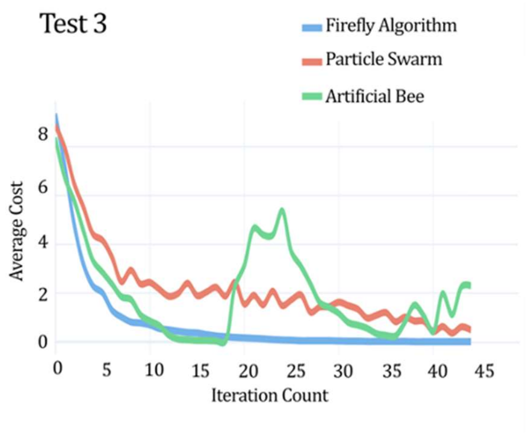

Figure 16. The average cost graph when there are 15 particles in the system and 45 iterations.

5. Conclusion

In this study, we developed an end-to-end generic crowd simulation framework in Unity 3D, which gives the developers and designers the ability to recreate, modify and test bio-inspired crowd simulation algorithms such as state-of-the-art firefly algorithm, particle swarm optimization, and artificial bee colony while enabling them to use those algorithms during game generation. As use cases, we generated three puzzle games using those three state-of-the-art algorithms within the framework. We introduced daisy attraction to the game as a side mechanic to manipulate the crowd behavior of the particles and used that behavior to create a puzzle video game. Then, we tested those algorithms' performances with different population sizes, iterations, and different field boundaries. The results show that the firefly algorithm (FA) outranks both the particle swarm optimization (PSO) and artificial bee colony (ABC) within the frame of the test. The tests show that FA achieves better outcomes in modeling the behaviors of small populated crowds when compared with ABC and PSO. However, with the larger field size, iterations, and population, the firefly algorithm falls behind ABC in terms of the best cost and best average cost, but the difference is so small that it does not affect the outcome of the simulation. When we evaluate the firefly algorithm in terms of computational speed, the firefly algorithm is not the best among these three optimization algorithms. However, if we are to consider the firefly algorithm's unique way of searching for global minimum and precision, we can understand that the firefly algorithm is open to many other improvements than daisy attraction since the particles in the firefly algorithm use most of the information in the system during their search for the global minimum, which makes the firefly algorithm a promising crowd simulation algorithm. The framework has the limitations of using only Ackley function as an objective function and generating only puzzle-like games without any additional coding. As future work, we plan to further develop our framework in order to simulate agent-based crowd behavior, to give the game developers and designers the ability to use different objective functions, to generate games from different genres, and to further examine the potential of the optimization algorithms on NPCs to mimic more human-like behavior during complex decision-making tasks.

6. References

[1] Lin, Y., Chen, Y., "Crowd control with swarm intelligence", *2007 IEEE Congress on Evolutionary Computation*, 2007, 3321-3328.

[2] Junaedi, H., Hariadi, M. and Purnama, I. K. E., "Multi agent with multi behavior based on particle swarm optimization (PSO) for crowd movement in fire evacuation", *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, 2013, 366-372.

[3] Mckenzie, F. D. et al., "Integrating crowd-behavior modelling into military simulation using game technology", *Simulation & Gaming*, 39 (1), 10-38, 2008.

[4] Yang, X. S., "Firefly algorithm, stochastic test functions and design optimization", *Journal of Bio Inspired Computation*, 2 (2), 78-84, 2010.

[5] Dey, N. (Ed.), "Applications of Firefly Algorithm and its Variants: Case Studies and New Developments", *Springer Nature*, 2020.

[6] Yu, T. et al., "Modelling and Simulation of Evacuation Based on Bat Algorithm", *IOP Conference Series: Earth and Environmental Science*, 267 (3), 2019, 032017.

[7] Wang, G. G. et al., "Monarch butterfly optimization", *Neural computing and applications*, 31 (7), 1995-2014, 2019.

[8] Darwish, A., "Bio-inspired computing: Algorithms review, deep analysis, and the scope of applications", *Future Computing and Informatics Journal*, 3(2), 231-246, 2018.

[9] Kowalski, P. A., et al., "On the use of nature inspired metaheuristic in computer game", *2017 Federated Conference on Computer Science and Information Systems*, 2017, 29-37.

[10] Díaz, G., Ilgesias, A., "Evolutionary Behavioral Design of Non-Player Characters in a FPS Video Game Through Particle Swarm Optimization", *13th International Conference on Software, Knowledge, Information Management and Applications*, 2019, 1-8.

[11] Ponticorvo, M. et al., "Approaches to Embed Bio-inspired Computational Algorithms in Educational and Serious Games", *CAID@ IJCAI*, 2017.

[12] Agarwal, S. et al., "Evaluation performance study of Firefly algorithm, particle swarm optimization and artificial bee colony algorithm for non-linear mathematical optimization functions", *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2013, 1-8.

[13] Ackley, D., "A connectionist machine for genetic hillclimbing", *Springer Science & Bussiness Media*, 28, 2012.

[14] Togelius, J. et al., "Search-based procedural content generation: A taxonomy and survey", *IEEE Transactions on Computation Intelligence and AI in Games*, 3 (3), 172-186, 2011.

[15] Unity Technologies., Unity 3d., http://unity3d.com/, Retrieved on July 21, 2020.

[16] Dented Pixel, LeanTween Assets, https://assetstore.unity.com/packages/tools/animation/leantween-3595, Retrieved on July 21, 2020.

[17] CraftPix, Assets, https://craftpix.net, Retrieved on July 21, 2020.

[18] Karaboga, D., Ozturk, C., "A novel clustering approach: Artificial Bee Colony (ABC) algorithm", *Applied Soft Computing.*, 11 (1), 652-657, 2011.