# Simulated World Research Report – Boids (Autonomous Agents)

## Autonomous Agents

“The term ***autonomous agent*** generally refers to an entity that makes its own choices about how to act in its environment without any influence from a leader or global plan” (Shiffman, D. 2012). Autonomous agents are those that, for all intents and purposes, make their own decisions. They do not receive ‘orders’ of any kind and simply follow their own set of rules to define their behaviour. Shiffman goes on to state in his book ‘The Nature of Code’ that autonomous agents have several key components that define their behaviour. They include, but are not limited to, “An autonomous agent has a *limited* ability to perceive environment”, “An autonomous agent processes the information from its environment and calculates an action”, and finally that “An autonomous agent has no leader” (Shiffman, D. 2012). From these three key components, we can begin to understand how exactly these autonomous agents function and operate. They have a limited ability to ‘see’ their environment, evaluate this information and process a response, whether that means going towards something or avoiding for example. Finally, these decisions are made solely by their self as they do not communicate with a leader.

“The behaviour of an agent is often determined dynamically based on its current perception of itself and the environment as well as a goal to achieve. This is the main difference between agents and conventional components” (Li, Q. Smith, G. 2016). Autonomous agents can be designed and shaped in many ways allowing for a huge scope of flexibility, from as little as a single behaviour or rule that defines their actions to a huge network of behaviours and relationships being the driving force behind them. Li and Smith also go on to state in their journal ‘Refining autonomous agents with declarative beliefs and desires’ that “an autonomous agent is one that is not only directed by its environment, but is also driven by internal motivation to achieve certain goals based on beliefs about the environmental behaviour”. They also go on to talk about how an agent’s beliefs and desires can help refine their actions and behaviours, and that by not only having the more linear ruleset, can lead to much more diversity within their behaviours and increasingly complex networks of agent interactions. “Agents with the same capability may have different goals and be deployed in different environments. Therefore, the design of such agents must not depend on following specific goals or be based on particular assumptions about the environment” (Li, Q. Smith, G. 2016). Such complicated systems are not always useful but it’s interesting to understand that autonomous agents ‘can’ be defined by so much more than just linear rulesets, and to an extent, can almost have their own ‘personality’ within a group.

“In the late 1980’s, computer scientist Craig Reynolds developed algorithmic steering behaviours for animated characters” (Shiffman, D. 2012). These behaviours allowed individual elements of a computer simulation to navigate and interact within their environment in a much more realistic manner than ever before. The simplicity of the rules, and the ability for each character to control themselves allowed for some surprising levels of complex behaviour. With the most famous example being “Reynold’s ‘boids’ model for ‘flocking/swarming’ behaviour” (Shiffman, D. 2012). This example follows the more traditional concept of less is more, with their behaviours being defined by a small ruleset and excludes some more modern approaches involving some of the concepts discussed above. At the time is was perhaps inconceivable to think of adding such complexities due to technological restrains, or simply that the initial concept has evolved over the years with more advanced versions coming into play. The fact remains though, that with such a simple set of defined behaviours, a huge diversity of patterns could be constructed by tweaking them.

## Background into Boids

The boids concept was originally developed By Craig Reynolds. The boids system uses an artificial intelligence algorithm designed to replicate the group behaviour of a flock of birds or school of fish for example. In the paper ‘Flocks, Herds, Schools: A Distributed Behavioural Model’ written by Craig Reynolds in 1987, the paper “explores an approach based on simulation as an alternative to scripting the paths of each bird individually”. Reynolds wanted to explore the idea that rather than each boid being scripted and following a given path, that they would have some form of limited AI, allowing for more freedom and a much more natural simulation. The boids system came about due to the realisation that “individual agents are becoming too complex to function as expected to” (Radwan et al. 2012), thus the ‘Multi Agent’ approach was born. This new approach allowed for much more flexibility and improved performance, “Distributing the system over multiple platforms and hardware enhances the robustness and reliability significantly” (Radwan et al. 2012). This was also boosted by the fact that the multi agent approach increased the computational efficiency by sharing the workload.

So, what is an intelligent agent, “An intelligent agent is a sophisticated entity that act autonomously by perceiving information from the environment using sensors, and decide to take action through reasoning process to perform its role and lastly act upon its environment through actuators” (Radwan et al. 2012).

Reynolds clearly had a fascination with birds, “The motion of a flock of birds is one of nature’s delights” (Reynolds C.W. 1987), and their almost synchronised group behaviour. These behaviours can also be seen in similar animal groupings, that Reynolds also states, including schools of fish and herds of land roaming animals. One of the most interesting components of a flock of birds is their “Strong impression of intentional, centralised control” (Reynolds C.W. 1987). Reynolds goes on to talk about the fact that a flock appears to have this form of centralized control but that the evidence shows that a flock’s motion is possibly the mere aggregated results of the actions of each individual bird with the flock. It was this conclusion and thought process that brought the concept of boids to life.

## What does it mean to be a Boid?

Following on from Reynolds theory it’s clear to see that he suggests that a flock’s motion is the culmination of each individual bird’s actions. So, to begin this concept Reynolds wanted to replicate the behaviour of each bird individually, instead of having a central control component. “For a bird to participate in a flock, it must have behaviours that allow it to coordinate with those of its flock mates” (Reynolds C.W. 1987). Reynolds decided upon three very simple behaviours, behaviours that most living creatures have to some extent. These behaviours are drawn upon by their needs, firstly their need to avoid a mid-air collision. Secondly their need to survive, to stick together is to survive. Birds are often not at the top of the food chain and as such having strength in numbers, is a key driving force behind their need to fly together as a single cohesive unit. Finally having so many eyes in the sky allows for a much larger search pattern with regards to searching out sources of food and spotting predators. With a couple of other advantages with regards to social interaction and mating also coming into to effect.

## How do they Work?

Now with a grounding into what they are, we can look at how. “In his 1999 paper ‘Steering Behaviours for Autonomous Characters,” Reynolds uses the word ‘Vehicle’ to describe his autonomous agents”. (Shiffman, D. 2012). Shiffman suggests that Reynolds took his inspiration from an Italian neuroscientist and cyberneticist ‘Valentino Braitenberg’. “in 1986, Valentino Braitenberg described a series of hypothetical vehicles with simple internal structures in his book ‘*Vehicles: Experiments in Synthetic Psychology’*. Braitenberg argues that his extraordinarily simple mechanical vehicles manifest behaviours such as fear, aggression, love, foresight and optimism. With Reynolds taking inspiration from Braitenberg” (Shiffman, D. 2012).

“Reynolds describes the motion of *idealized* vehicles (idealized because we are not concerned with the actual engineering of such vehicles, but simply assume that they exist and will respond to our rules) as a series of three layers - Action Selection, Steering and Locomotion” (Shiffman, D. 2012).

The first layer Reynolds discusses is ‘Action Selection’ and refers to that vehicles goal or goals. The Vehicle is also free to select an action or combination of actions to achieve these goals. The vehicle is able to perceive its local environment and decide upon an action based upon its desire.

Shiffman uses the example of a person who can see a zombie approaching them, the person does not wish to get eaten and as such steers away from the zombie and moves away. “Reynolds paper describes many goals and associated actions such as: seek a target, avoid an obstacle and follow a path” (Shiffman, D. 2012). This represents ‘Action Selection’ and involves strategy, goals and planning to achieve the desired outcome.

Now that the vehicle has a goal, or set of, its free to select an action, or combination of, based on the goal. This is the second layer and involves the vehicle calculating its next move, this involves using Reynolds steering force formula ‘***steering force = desired velocity – current velocity****’*. Whereby the vehicle calculates it steering force based on its desired velocity, a combination of its actions, and then subtracting that from its current velocity creating a steering force that equates to the combination of the action(s) it chose to take earlier.

The third and final step refers to Locomotion, the animation or articulation of the vehicle in question. “Locomotion is the bottom of the three-level behavioural hierarchy described above. The locomotion layer represents a character’s *embodiment*. It converts control signals from the *steering* layer into motion of the characters ‘body’” (Reynolds, C.W. 1999). This final layer doesn’t alter behaviour in any way, but rather involves the animation of the vehicle or character and how they respond based on the actions from the previous two layers.

## The Golden Rules of Boids (Flocking)

## Separation

“Separation: steer to avoid crowding local flockmates” (Reynolds, C.W. 2001). Separation refers to a boids will to avoid the space occupied by other local boids. “steering behaviour gives a character the ability to maintain a certain separation distance from other nearby” (Reynolds, C.W. 1999). This is useful to avoid overcrowding and can be increased or decreased based on a specified distance.

To calculate this behaviour, a ‘neighbourhood’ check is made, with a certain distance used to represent the boids ‘sight’ to calculate which boids are deemed neighbours. These neighbours are then checked against to see just how close they are. If any are too close, less that the ‘min neighbour distance’, “a repulsive force is computed by subtracting the positions of our character and the nearby character, normalising, and then applying a weight” (Reynolds, C.W. 1999).

## Alignment

“Alignment: steer towards the average heading of local flock mates” (Reynolds, C.W. 2001). Alignment refers to the accumulated direction of all neighbouring boids in relation to a given boids current heading, to calculate this value the boid must first identify its neighbours. Again a ‘neighbourhood’ check is made, with a certain distance used to represent the boids ‘sight’ to calculate which boids are again deemed neighbours. These ‘neighbours’ combined velocities are then added together and averaged out by the number of neighbours, thus creating a new average velocity or heading of the group.

“Alignment can be computed by finding all characters in the local neighbourhood, and averaging together the velocity. This average is the ‘Desired Velocity’, and so the steering vector is the difference between the average and our character’s current velocity” (Reynolds, C.W. 1999).

The steering behaviour this rule creates will lead the boid to turning and facing the same direction as its neighbours or ‘flock’, and thus head in the same general direction

## Cohesion

“Cohesion: steer to move towards the average position of local flockmates” (Reynolds, C.W. 2001). Cohesion represent a given boids ‘desire’ to form a group, to cohere and attracts agents pulling them in. Once again to begin a calculation for this behaviour, the ‘neighbourhood’ check needs to take place first, a central position within the given boids neighbours is the identified by adding all the positions together and averaging them out, thus creating our cohesion position. Reynolds relates this behaviour to that of a ‘magnet’ or ‘center of gravity’, pulling in the boid.

## Seek

Following on from cohesion is seek, seek used the position provided by Cohesion to form the attraction and can be calculated like so - ‘***Desired velocity = normalize (boid position – target positon’***. The ‘target’ is provided by cohesion, seek simply uses this information to calculate a given boids ‘desired velocity’ i.e. the position it wants to head towards, or goal.

## Other Behaviours

In Reynolds journal ‘Steering Behaviours for Autonomous Characters’ he discusses several other behaviours. These behaviours can be used to enhance the boids knowledge and ability to perceive its environment, and some of note include:

### Flee

Flee is one such example, “Flee is simply the inverse of ***Seek*** and acts to steer the character so that its velocity is radially aligned away from the target. The ***Desired Velocity*** points in the opposite direction” (Reynolds, C.W. 1999).

As Reynolds states this behaviour is simply an inverse of seek, and instead of acting as a magnet or center of gravity, it acts as a repelling force, like pushing the same poles of a magnet together.

### Pursuit

Pursuit acts in a similar manner to that of seek with one key difference, instead of heading towards a current positon, it moves towards a predicted position based on where it is currently and its velocity. “Pursuit is similar to seek except that the target is another moving character. Effective pursuit requires a prediction of the target’s future position” (Reynolds, C.W. 1999).

### Obstacle Avoidance

This behaviour is all about a given boids ability to, you guessed it, avoid obstacles. This behaviour can be seen as similar to ***Flee*** but with one key difference, flee will cause the boid to turn away regardless of the position of the other party relative to its own position and heading. Obstacle avoidance is slightly more advance in that it only triggers a response from a boid if that party lies directly in its path. Reynolds describes a scenario involving a car driving along a road, parallel to a wall. No response is triggered as the wall is not directly in its path. If the car had a flee behaviour it would pull away from the wall regardless of its position in relation to its forward-facing direction.

“There is an important distinction between obstacle avoidance and flee behaviour. Flee will always cause a character to steer away from a given location, whereas obstacle avoidance takes action only when a nearby obstacle lies directly in front of the character” (Reynolds, C.W. 1999).

### Path Finding

“Path following behaviour enables a character to steer a predetermined path, such as a roadway, corridor or tunnel. This is distinct from constraining a vehicle rigidly to a path like a train rolling along a track. Rather path following behaviour is intended to produce motion such as people moving down a corridor” (Reynolds, C.W. 1999). Path following behaviour can be achieved by creating a path, Reynolds goes on to say in his paper ‘Steering Behaviours for Autonomous Characters’ and that the goal of such a behaviour is to move a character along said path while staying within a certain distance from the center. Or potentially by creating a set of waypoints for the agent to follow, with each agent moving towards a given waypoint, and then when in range, i.e. close enough, their waypoint target is updated pointing towards a new one and thus a path following behaviour is created.

## Combining Behaviours

By mixing and matching individual behaviours hugely varied and diverse systems can be created and serve as building blocks to create much more specialised patterns and behaviours. For the creation of realistic and believable patterns to emerge, these behaviours need to be finely blended together to create the correct balance and desired outcome. One of the simplest ways to combine behaviours is by weighting. Each behaviour creates some form of value or output, and by multiplying the ‘importance’ of these values together can lead to some incredibly diverse patterns and wild changes with even the smallest adjustments or changes.

Reynolds on behaviours - “They are components of a larger structure, like notes of a melody or words of a story” (Reynolds, C.W. 1999).

## Real World Examples

The boids flocking model has many applications, from large scale battles, animal flocking behaviour and may more. The Model has been used on many films to simulate many different kinds of behaviours, some well-known examples include:

Batman Returns (1992) - Bat Swarms & armies of penguins “The 1992 Tim Burton’s movie, presented bat swarms and penguin flocks created using modified versions of the original boids developed at Symbolics” (Nunes de Castro, L. 2006).

The Lion King (1994) - Simba Stampede Scene, during the film the main character, a lion named Simba, enters a valley after being tricked by his uncle, scar. Hyenas herd wilder beast down into the valley. The wilder beast were being controlled by a similar system to boids.

The Matrix Revolutions (2003) - Attack on Zion, During the climax of the final film in the series, hundreds of thousands of Sentinels swarm into the docking area of Zion, the last human city. A large-scale battle ensues, with the sentinels again using a system not to dissimilar to Craig Reynolds boids.

## Implementation

### Overview

To implement the boids system several key issues needed to be addressed before even a single line of code could be written. The first was the concept of trying to include a boids system into a game pipeline so that it could potentially be used in a larger project or game. The second was how the system would be structured and who and what would be responsible for certain tasks and actions.

It was important to create the entire system with through the boid manager, as that has allowed maximum flexibility when it comes to simply ‘plugging in’ and using the system. If game.cpp has a large no of objects to handle itself, it would of lead to a much more complicated system in terms of ease of use. Currently the whole system can be created by simply constructing a new BoidManager object and allowing that to function as the mastermind behind the entire system.

### Structure

To begin thinking about code structure my initial thought was immediately to have a manager system in place that acted as an ‘overseer’ for the boids and the system itself. It would be responsible for not only the creation of the boids, but when they were activated and deactivated, when they were drawn and what they could do and when could they do it. By having such a system in place, it also made it much easier to control access with the boid manager now the ‘goto’ object to get a handle on anything boid related.

For the boids themselves I initially followed guidance from David Shiffman’s processing example, in his structure he allows each boid to have its very own version of each behaviour. This is something that was simple to get things started and working, but not something I was keen of following later into development. A more elegant system was developed with the boid manager now having a vector of behaviours. This allowed for only one ‘master’ version of each behaviour needing to be active in the system. The behaviours are then fed into each boid when they are needed giving the boids access to the behaviour and allow them to update their selves, without the need of each boid having their own copy or version of the behaviours taking up extra space in the system.

Another key feature was the implementation of a custom data container, named ‘BoidData’. This container would contain information on all of the boids variables, including their speed, how far they can see, separation, and of course all the weighting for their behaviours. This again allowed for less variables per boid with each boid having their own copy, but instead referring to the variables contained within BoidData.

### System Features - Boid Specific

The boids each have their three base behaviours, alignment, separation and cohesion. These allowed the boids basics of flocking behaviour with plenty of scope for further advancements and improvements. With just a few of the behaviours explored earlier out of the many, many available, it was clear that not all of them would be able to make it into the final system, so instead focus was put towards a select few that were deemed more interesting and unique.

One of the first mechanics to be added was that of factions, and an accompanying behaviour, flight/fight. By allowing multiple factions led to several new simulation options, specific faction boids could either have an attraction state towards other factions boids, simply ignore the fact that they were of a different faction entirely, or simply run on sight.

Another early additional feature implemented was that of the player character. With the addition of the player came that of an additional behaviour, predator avoidance. This new behaviour worked both ways, as if the value was weighted positively the boids would avoid the player based on whether or not they could see them, and when set to a negative weighting they would become attracted, again based of if the boid could indeed see the player. This was a very interesting addition and lead to some interesting combinations of behaviours. The ‘Bat Swarm’ example, mentioned above, was now possible, with the player character now able to attract a cluster or swarm or boids around them.

An interesting example would be having one faction able to see the player from a large distance with an attract like behaviour, and another that could also see the player with the same state, with the one key difference being that the second faction wanted to avoid other factions boids via a flight/fight weighting.

The faction system soon followed with three different factions, red green and purple. Each faction had the same variables that could be changed independently of one another thanks to the BoidData concept implemented earlier, the boid manager now has three separate versions of BoidData with one assigned to each, allowing for even more complicated behaviour combinations to be created.

Another behaviour that was of particular interest was that of path finding. Alongside the features mentioned above this one allowed to some really spectacular combinations of behaviours to be created. During runtime waypoints are able to be moved and adjusted on the fly, allowing different shapes and patterns to appear amongst the boids. This was a more difficult behaviour to implement in the beginning and issues arose as to how exactly the boid would know which waypoint is was currently going to if that position was constantly changing?

The solution, found after a few failed attempts, was rather simple and elegant. Instead of giving the boid an actual vector or position to go too I instead just gave them a waypoint id, an integer that told the boid which waypoint they were currently going to. This id was then used to check the container, holding the waypoint positions, and pull out the exact position it was heading to. This meant that waypoint positions could be changed dynamically during runtime and allow the boid to maintain its target regardless of where it was in the world. This is something that likely features a lot in industry, with enemies chasing the player as they move around the game, or cars in racing games knowing where others are being a couple of examples.

### System Features - Non Boid Related

### FileReader

As an additional feature and with the idea that the system could be used by a no programmer, text file loading was added, with the ability to change values in a simple text file, this allows for behaviours, and combinations of, to be created without the need to change or enter any code.

### AntTweakBar

Another key feature that was really needed was one that allowed the user to change variables dynamically during runtime. The easiest concept would have been to map the keyboard to certain functions that changed parameters. This would likely be a band aid fix and one that wouldn’t have been ideal. Thankfully AntTweakBar came to the rescue. This neat package allowed easy access to the variables that controlled the simulation during runtime. “AntTweakBar is a small and easy to use c/C++ library that allows programmers to quickly add a light and intuitive graphical user interface into graphic applications” (Decaudin, P. et al. 2013).

## Performance enhancement

Being a system involving a large number of autonomous agent it was clear there would be a huge number of calculations going on at any given time within the system. There are undoubtedly many ways to optimise an application but two that stood out as possibilities were special partitioning and a limited group update system.

### Spatial Partitioning

“Efficiently locate objects by storing them in a data structure organizes by their positions” (Nystrom, R. 2011). As Nystrom suggests in his book ‘Game Programming Patterns’ the key concept to spatial partitioning is organisation. By dividing up the world, or game area, in to smaller more manageable chunks it can become a lot more manageable allowing for quicker updates and much more optimised performance, potentially.

“For a set of objects, each has a position in space. Store them in a spatial data structure that organizes the objects by their positions” Nystrom, R. (2011). By doing something similar to this, each time a boids position is checked against its neighbours, it will only check those that are in the same area or location, rather than checking each and every other object in the system. A link list would clearly be a good choice for this as it’s very easy to keep track of what is where and update the objects accordingly.

Although a sound concept and one that is, in most case scenarios likely to aid performance drastically especially on larger more complex systems, the possibility still remains that all boids or objects could reside in the same area or zone, thus causing slowdown and performance issues. In many systems, this would unlikely be a problem but the fact remains, and indeed the possibility that it could happen, is ever looming.

Although in theory a relatively simple concept to grasp, and one that clearly had many more pro’s to con’s, it wasn’t one that was implemented into the final system, this was due to several reasons, the main being the time restraint on creating it and the second my ability to put theory into practice within the time available.

### Group Update

Being that spatial partitioning was just out of reach, a more simplistic concept materialised in the form of a group update mechanic. What it allowed for was a select number of boids to be ‘updated’ at any given time. The system would then cycle through to the next ‘group’ of boids and update them, going on in a constant loop. Although some performance may have been gained its unlikely to be able to match the potential performance of spatial partitioning for one very obvious reason. That being the fact that although only a select number of boids were being checked at a time, they were still being checked against all the other boids in the system. So, although it may create some small gain on a relatively small simulation of boids, this gain would undoubtedly begin to diminish the larger the simulation becomes, whereas special partitioning would experience gains at any level simulation, and perhaps in an ideal world or system both could be implemented in tandem to create an even more optimised system.

### Performance summary

In conclusion performance wasn’t the main focus of the system implemented, with behaviours being the key points of interest and how to keep everything organised and tidy within the codebase. It is however something of interest and something that, in future may take more of a precedence.

## Evaluation

Overall the structure of the system implemented was something that was surprisingly well implemented, with the behaviours and boid data a particular highlight. Alongside that was the path finding system that is a joy to play around with, allowing for some delightful interactions between factions and providing a pleasing experience and simulation.

By adding the ability to change variables via a text document was also another feature that turned out very well, with an unusual concept being used. That being the boid manager having a ‘FileReader’ object that it passed through to the BoidData constructor. The BoidData would then pass itself through to the FileReader to unpack its variables from a text file, this was something that was very interesting to implement and a joy to see working.

Something that was attempted was the idea of each boid owning its own VBGO, or model. In the current system, the boid ‘is’ the VBGO. Having tried to change this setup lead to more pointers being needed and needing to be called for certain behaviour checks ect, this unfortunately cause huge performance drops, even in release, and as such needed to be reverted.

Anther failed concept was, not a change to the boid system itself, but the game loop itself and how it rendered what needed to be drawn at any given time. The concept was simple, to have a ‘GameStateManager’ class that was able to change ‘scene’ and thus what needed to be drawn based on what state the game was in. This concept was unfortunately scrapped after it was found to be incompatible with the current DirectX setup for drawing in the solution, with several components outside of game relating to rendering it was too large a task to undertake with the limited time left, and the idea was therefore binned.

The addition of the AntTweakBar was huge, as being able to modify values during runtime was key to creating a fluid and dynamic user experience. These variables could also be changed in house with non-programmers such as artists or designers now able to create complex combination of behaviours without the need to access any code at all.

Following on another feature that would have been nice would be that of the ability to create text files containing new sets of behaviours. That way when a complex set of behaviours had been created the user could simply click a button and have the program create a new text file with the new variables. If the system were to be advance this would undoubtedly be near the top of the priority list.

If the system was to be advanced in the future the first point of call would undoubtedly be that of performance and optimisation, with some form of special partitioning top of the list. This would allow for a lager, smoother simulation and system to be active with many more boids on screen at any given time.

Other advancements would probably be in the behaviours category, adding in more complicated behaviours would be interesting, adding actual sight to the boids, see ahead of their self, rather than the current system whereby they are deemed in sight at any angle. This would potentially open up the door for actual flocking patterns to appear and something that would be very interesting to explore and evolve.

References:

1. Radwan, M. Badr, A. Farag, I. (2012). Task Allocation in Distributed Artificial Intelligence using Boids Model. *International Journal of Computer Applications*. Volume 53(2), pp 40-46.
2. Reynolds, C.W. (1987). Flocks, Herds and Schools: A distributed Behavioural Model. *Computer Graphics*. Volume 21(4). Pp 25-34.
3. Reynolds, C.W. (2001). *Boids: Background and Update*. Available from <http://www.red3d.com/cwr/boids/> [Accessed 06/02/2017].
4. Shiffman, D. (No Date). *Flocking*. Available from: <https://processing.org/examples/flocking.html> Accessed 07/02/2017].
5. Coding Train. (2015). *The Nature of Code – series*. Available from: <https://www.youtube.com/watch?v=JIz2L4tn5kM&list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvlW> [Accessed 07/02/2017].
6. Shiffman, D. (2012). Autonomous Agents. In: Shiffman, D., (2012). *The Nature of Code: Simulating Natural Systems with Processing*. Unknown: The Nature of Code, pp. 260-322.
7. Li, Q. Smith, G. (2017). Refining autonomous agents with declarative beliefs and desires. *Formal Aspects of Computing.* Volume 29, pp 227-249.
8. Reynolds, C.W. (1999). Steering Behaviours for Autonomous Characters. [Accessed 15/03/2017].
9. Nunes de Castro, L. (2006). *Fundamentals of Natural Computing.* Basic Concepts, Algorithms, and Applications. Chapman and Hall/CRC.
10. Nystrom, R. (2011) *Game Programming Patterns***.** Apress.
11. Decaudin, P. et al (2013). AntTweakBar. Available From: <http://anttweakbar.sourceforge.net/doc/> [Accessed 03/03/2017].