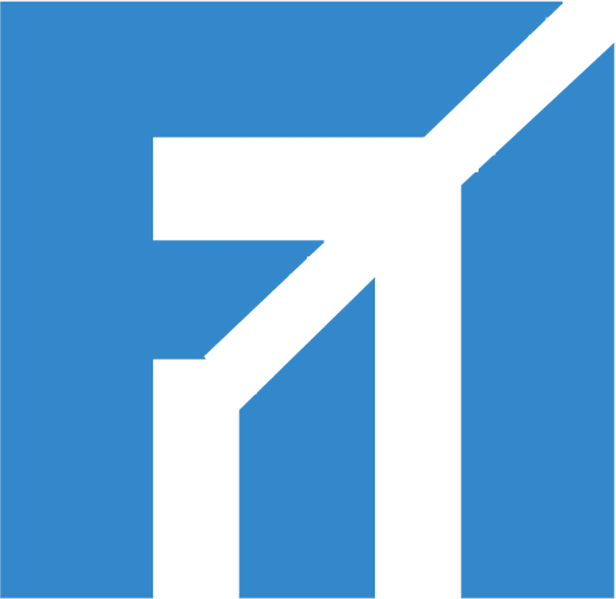
**“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI**

**FACULTY OF COMPUTER SCIENCE**



BATCHELOR THESIS

**Application of K-means Clustering in Video Games**

Proposed by

**Andrei Ghiran**

**Session:** 06.2020

Scientific coordinator

**Prof.lect.dr. Moruz Alex**

**“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI**

**FACULTY OF COMPUTER SCIENCE**

**Application of K-means Clustering in Video Games**

**Andrei Ghiran**

**Session:** 06.2020

Scientific coordinator

**Prof.lect.dr. Moruz Alex**

SUMMARY

Introduction 3

1.Similar Applications 4

1.1. Deep learning agents 4

2.2. Computer vision-based agents 5

2. Implementation 7

2.1. AI adversary 7

2.1.1 The K-means Algorithm 7

3. User Manual

4. Conclusion and Future Goals

INTRODUCTION

From the day I started my studies at the faculty of computer science I knew I wanted to pursue a career in game developing. To be one step closer to my career of choice I decided that for my bachelor thesis I would combine my passion for game developing and the limitless applications of machine learning. Using the Unity engine I developed a simple game that showcases the K-means algorithm by using it to train an Artificial Intelligence adversary for the player.

The game has a simple premise, the player controls a blue square that can only move on a grid in the 4 cardinal directions and must navigate around walls and reach the goal, the AI adversary, represented by a red square, with the same movement constraints as the player and only moving at the same time as the player, must intercept and bump into them to end the game. The adversary doesn’t know the position of the player at all times, he has a vision radius in witch he can detect the player, acquire his position and save it in a file for later use. The K-means algorithm uses the saved player locations to form clusters and determine an advantageous position from where the adversary can intercept the player, the using the A star algorithm the adversary will find a path to this more advantageous position and follow it.

**1.SIMILAR APPLICATIONS**

Information on the use of machine learning techniques in video games is mostly known known publicly through research projects as most gaming companies don’t publish specific information about their intellectual property. The most common use of clustering algorithms in the domain of video games is the classification of player behaviour in massive multiplayer online games (MMOs). These behaviours range from completing missions to exploring the game wold and even spending time with other payers. The classification is used by the developers to understand what most players enjoy doing the game and this knowledge influences the further development of the game. K-means clustering is also commonly used for procedural content generation.

The most popular application of machine learning in games is the use of deep learning agents that compete with professional human players. Another popular type of agent is the computer vision-based AI player. The most significant application of machine learning has been done on games such as Dota 2, the StarCraft series, Pong and Doom. Games that did not originate as video games such as Chess and Go have also been subject to machine learning research.

**1.1. Deep Learning Agents**

Deep learning focuses heavily on the use of artificial neural networks that learn to solve complex tasks. Deep learning uses multiple layers of artificial neural networks and other techniques to progressively extract information from an input. Due to this complex layered approach, deep learning models often require powerful machines to train and run on.

**Chess** is considered a difficult AI problem due to the computational complexity of its board space, its state space being 10^120 possible board states. Similar strategy games are often solved with some form of a Minimax Tree Search. These AI agents have been able to beat professional human players, such as the historic 1997 Deep Blue versus Garry Kasparov match. Since then, machine learning agents have only shown even greater success.

**Go** poses an even more difficult AI problem then chess. The state space of Go is approximately 10^170 possible board states, much greater then the state space of chess.

Google’s 2015 AlphaGo was the first AI agent to beat a professional Go player. It used a deep learning model to train the weights of a Monte Carlo tree search. The deep learning model consisted of 2 Artificial Neural Networks, one network to predict the probability of potential moves made by the opponent and the other network to predict the win chance of a given state. The combination of the deep learning model and the Monte Carlo tree search allowed the agent to explore the state tree much more efficiently than if it was using just the Monte Carlo tree search.

AlphaGo was initially trained on games against human players and later on games against itself. Later, in 2017, another implementation of AlphaGo, AlphaGoZero, was made public. This implementation was able to entirely train by playing against itself and it would train much faster then its predecessor. In the final stages of its training AlphaGoZero would develop new advanced strategies that, until then, were never used by a Go player.

**The StarCraft series** is a series of real-time strategy video games that have become very popular environments for AI research. Blizzard, the game’s developer, and DeepMind have collaborated to release a public StarCraft 2 environment for AI research.

Alphastar was the first AI agent to beat a professional StarCraft 2 player without any in-game advantages. Initially for its deep learning network the agent used a simplified zoomed out version of the gamestate as imput, but later it was updated to play using a camera like other human players. Up to the date of writing this document the code or architecture of the model have not been publicly released by the developers of Alphastar, but they have listed several machine learning techniques that they used, such as relational deep reinforcement learning and long short-term memory.

Initially Alphastar was trained with supervised learning, it watched replays of many human games in order to learn basic strategies, later it trained against different versions of itself an was improved trough reinforcement learning. The final version was successful but it was only trained to play on a specific map and matchup.

**Dota 2** is a multiplayer online battle arena (MOBA) game. Because of its complexity traditional AI agents have not been able to play at the same level as professional human players. The only widely published information on AI agents attempted on Dota 2 is OpenAI’s deep learning Five agent.

OpenAI Five utilized separate Long Shor-Term Memory networks to learn each hero in the game. It trained using a reinforcement learning technique known as Proximal Policy Learning. It trained against itself on a system containing 256 GPUs and 128,000 CPU cores accumulating 180 years of game experience each day. OpenAI Five's first public appearance occurred in 2017, it won a one-on-one game against a professional Dota 2 player known as Dendi. The following year the agent advanced so much that it could performe as a team and in a 2019 series of games beat the 2018 Dota 2 champion team.

**1.2. Computer vision-based agents**

Computer vision focuses on training agents to gain a high-level understanding on digital images and videos. Many computer vison techniques incorporate forms of machine learning and have been applied to various video games. This technique focuses on interpreting game events using visual data, in some cases AI agents, using model-free techniques, have learned to play games without any direct connection to internal game logic, solely using video data as input.

**Pong** is a table tennis sports game featuring simple two-dimensional graphics, manufactured by Atari and originally released in 1972. Andrej Karpathy, Tesla's director of artificial intelligence, demonstrated that a neural network with just one hidden layer is capable of being trained to play Pong based solely on screen data.

**Doom** (1993) is a first person shooter (FPS) game. A group of student researchers from Carnegie Mellon University used computer vision techniques to create an agent that could play the game using only image pixel input from the game. The students used convolutional neural network layers to interpret image data and output valid information to a recurrent neural network which was responsible for outputting game moves.

Other uses of machine learning in video games focus more on generating procedural content rather than AI agents that interact with the player. Procedural content generation is the process of generating data algorithmically rather than manually. Examples of procedurally generated content can be found in the weapons of Borderlands 2, the map layouts of Minecraft and the entire universes in No Man’s Sky. Common approaches to procedural generation include techniques that involve grammars, search-based algorithms, and logic programing. These approaches require a human to manually the range of content possible, but machine learning is theoretically capable of learning the features of possible content when given examples to train on, thus reducing the time spent on defining the range of desired content. Machine learning techniques used for content generation include Long Short-Term Memory, Recurrent Neural Networks, Generative Adversarial Networks and K-means clustering.

**Galactic Arms Race** is a space shooter video game that uses neuro-evolution powered procedural content generation to generate unique weapons for the player, based on his personal preferences. This game was a finalist in the 2010 Indie Game Challenge and its related research paper won the Best Paper Award at the 2009 IEEE Conference on Computational Intelligence and Games. The content was generated using a form of neuro-evolution called cgNEAT.

Machine learning is not a new addition to the domain of video games, but that does not mean there is no room for new and innovative application of its techniques. Over the years these techniques have continued to evolve and will continue to do so with the development of new video games and technologies.

**2. IMPLEMENTATION**

The project had been developed using the Unity engine, a cross-platform platform game engine developed by Unity Technologies. The engine can be used to create three-dimensional, two-dimensional, virtual reality and augmented reality games as well as simulations, animations and other experiences. Several major versions of unity have been released since its launch. As of writing this document the latest stable version in 2019.4.0, the project has been developed with version 2019.3.4f1.

The project can be divided in three integral components: the AI adversary, the player characters and the environment.

**2.1** **Used Technologies**

**2.1.1 Unity Game Engine**

TheUnity game engine is a cross-platform platform game engine developed by Unity Technologies. The engine can be used to create three-dimensional, two-dimensional, virtual reality and augmented reality games as well as simulations, animations and other experiences. Several major versions of unity have been released since its launch. As of writing this document the latest stable version in 2019.4.0, the project has been developed with version 2019.3.4f1.

The Unity development environment designed to be easy to understand by users without programing knowledge. Users add GameObjects into a scene and can add components to those objects. Component define how a GameObject behaves in the scene.

**2.1.2 C# Programing Language**

C# (pronounced see sharp) is a modern, object-oriented and type-safe programing language. Its features aid in the construction of robust and durable applications. Garbage Collection automatically reclaims memory occupied by unused or unreachable objects, The type-safe design of the language makes it impossible to read from uninitialized variables, to index arrays beyond bounds or to perform unchecked type casts;

**2.1.3 C# Built-In types**

The C# programing language offers a set of primitive types of variables for storing simple data types.

* **bool** represents a Boolean value which can be either true or false. To perform logical operations with values of the bool type, use Boolean logical operators such as “!” (not), “&&” (and), “||” (or). The bool type is the result type of comparison and equality operators (greater >, lower <, greater or equal >=, lower or equal <=, equal ==).
* **int** (integral numeric type)represents integral numbers, they support arithmetic, bitwise logical, comparison, and equality operators. Values for his type range from -2,147,483,648 to 2,147,483,647.
* **float** (floating-point numeric type) represents real numbers and support arithmetic, comparison, and equality operators. This type has a precision of ~6-9 digits.
* **double** is also a floating-point numeric type, but it is more precise that the float type, having a precision of ~15-17 digits.
* **string** represents a sequence of zero or more Unicode characters. The equality operators **==** and **!=** are used to compare values of string objects. The **+** operator is used to concatenate multiple strings together (“apple ” + “pie” results into “apple pie”). The **[]** operator is used for read only access to individual characters of a string by the index of their position. Valid index values start at 0, meaning the first character of a string has the index 0;

**2.1.4. C# Code Statements**

C#, like many other programing languages, supports the use of complex code statements used for modeling program behaviour.

* **If-else** statements identify which code branch to run based on the value of a Boolean expression. The if statement evaluates a Boolean expression and if the expression is true then the code section inside the first branch of the if is executed, if the expression is false then the code section inside the second branch of the if is executed. In the absence of an else branch if the condition is false no code will be executed.
* **Switch-case** statements are similar to if-else statements, as in they are used for decision making based on an expression. The switch-case statement evaluates an expression and will execute the code sample in the corresponding case.
* **For and While** ……

**2.2. AI adversary**

The adversary is represented by a red square

The core components of the AI are the K-means clustering algorithm for determining an advantageous position and the A star algorithm for calculating a path to that position.

**2.2.1. The K-means clustering algorithm**

Clustering is used for partitioning a set of objects into groups, called clusters. Objects that are similar, according to a certain similarity measure, are places in the same cluster, while dissimilar objects are placed in different groups.

The K-means algorithm aims to partition n objects , in k () clusters , in witch every object belongs to the nearest cluster center. The most common algorithm uses an iterative refinement technique.

Given an initial set of k means , represented by centroids corresponding to each clusters, the algorithm alternates between two steps:

**Assignment:** The algorithm assigns each object to the cluster with the nearest mean, that with the minimum Euclidean distance between the object and the centroid corresponding to that cluster. If an object has the same minimum Euclidean distance to multiple centroids, it will be randomly assigned to one of the cluster that correspond to those centroids.

, where t represents the current iteration.

**Update:** Recalculate the means (the position of the centroids) for the new clusters.

The algorithm stops when, after 2 consecutive iterations the assignments no longer change. The algorithm does not guarantee to find the global optimum partitioning, the result may depend on the initial clusters. As the algorithm is usually fast, it is possible to run it multiple times and obtain different clusters.

The algorithm is most commonly used to assign objects to the nearest cluster by Euclidean distance. Using a different distance function may result in the algorithm never stopping.

Methods commonly used for initializing the clusters are **Forgy** and **Random Partition**. The Forgy method randomly choses k objects from the data set and uses these as the initial means. The Random Partition method first randomly partitions all the objects to one of the clusters and then proceeds to the update step, computing the initial means to be the centroids of the clusters randomly assigned points.

**Illustrated example of the K-means algorithm:**

Given a set of objects, represented by the squares in Figure 2.2.1.1**,** we apply the k-means algorithm to partition them in k = 3 clusters.

**Step 1**: k initial means, represented by the colored circles are randomly placed within the data domain. (*Figure 2.2.1.1*);

**Step 2**: k clusters are created by assigning each object to the closest mean. (*Figure 2.1.1.2*);

**Step 3**: the cluster centroids become the new means (*Figure 2.2.1.3*)

**Step 4:** steps 2 and 3 are repeated until the cluster assignments don’t change anymore (*Figure 2.1.1.4*);

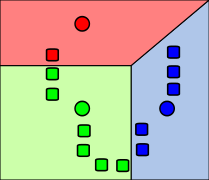
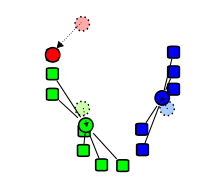
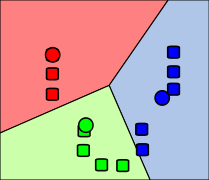
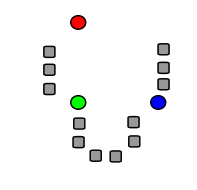
****

Figure 2.1.1.1

Figure 2.1.1.2

Figure 2.1.1.4

Figure 2.1.1.3

In the project the k-means algorithm is used to partition the position where the AI adversary has encountered the player in previous sessions and calculate an advantageous position to were the AI can move to intercept the player.

**Point Saving:**

The AI has a vision field in which it can detect the player character. The vision field is implemented by using a Box Collider 2D component from the unity engine. Box Collider 2D components are invisible rectangle shapes used to handle physical collisions between game objects. This Box Collider 2D component has the attribute “Is Trigger” activated with allows it to not collide with collider component of other game objects, instead it will act as a trigger calling a function whenever another collider enters it. A function encapsulates a section of code witch you can use multiple times in your code just by calling the name of the function. Every time the player character enters the vision collider the function that is called records the players coordinates in a List object of Vector2 objects named “positions”. A list object represents an ordered collection of objects of the same type, the order of the objects is determined by the order in witch they have been added to the list. To initialize a list you write New List<T>() method where T is the typo of the objects you want to store in the list. The objects in a list can be accessed by writing ListName[i], where i is an index witch starts at 0. To add objects to the list you use the ListName.Add(Object) method and to get the number of objects in the list you use the ListName.Count property. A Vector2 objects is an object that stores two dimensional coordinates, they can be accessed by calling the properties Vector2.x and Vector2.y. When the user closes the application a function called “OnApplicationQuit()” is called witch contains a function called “Save()” in witch using a for statement we iterate trough all the points in the positions list and write them in a file called “points.txt”, witch can be found at the application’s location, using a StreamWriter object. A for is used to execute a section of code multiple times in a loop, you can control the number of executions with a Boolean expression. A for statement can look like “for( int i = 0; i < 5; i++){\*section of code\*}” the integer I will start as 0 and after each execution of the code section it will be incremented by 1, i++ is equivalent to adding 1 to I after he execution of the code, when I is no longer lower than five the for loop will stop. A StreamWriter object is used to write a sequence of characters in a particular encoding. We create a StreamWriter by calling one of its constructor, StreamWriter(string path, bool append), where path represents the complete path to the file you want to write in and append is true if you want to write data at the end of the file or false if you want to overwrite the data in the file. Id the file does not exist at the path specified then the constructor will create the file at that path. The coordinates are written in pairs divided by an empty space, on separate lines.

**Point Loading:**

In the function “Start()” witch is called once, when the application starts, we initialize five list:, points\_go used for storing game objects corresponding to the point we will load, positions used for storing Vector2 objects corresponding to the new positions we will save when the player enters the vision field, old\_points used for storing all the Vector2 objects corresponding to the coordinates we will load from the “points.txt”, cluster\_centers used for storing game objects corresponding to the cluster centroids we will use in the k-means algoritm, clusters used for storing lists of game objects witch correspond to the clusters used in the k-means algorithm. Here we call the function “Load()” in witch we read the coordinate from “points.txt”, using a StreamReader, and place them into the old\_points list. A StreamReader object is used to read characters from a byte stream, in our case the byte stream comes from the “points.txt”

.

**3. USER MANUAL**

After starting the application you must chose one of two modes, Play Mode or Developer Mode, by pressing one of the corresponding buttons.

**Play Mode** is intended to exhibit how the game would behave in a normal play session. All the computing is done in the background. The player controls the blue square and the goal is to reach the green area by navigating the grey maze and avoiding the AI adversary, represented by the red square.

**Controls:**

* Use the arrow keys to move the player character (blue square). You can only move in a single direction at a time.
* Use the “Escape” key to close the application.

**Developer Mode** is intended to exhibit the internal mechanics of the game. When the session starts in developer mode the AI does not run the k-means algorithm right away and it will not do so until a specific key is pressed. In this mode there is a list of important keyboard inputs.

**Controls:**

* Use the arrow keys to move the player character (blue square). You can only move in a single direction at a time.
* Press “D” to display a graphical representation of useful information.
  + The white points represent all the positions recorded and used by the k-means algorithm;
  + The colored triangles represent the means of the clusters;
  + The dark red diamond represents the AI’s navigation goal;
  + The small red squares represent the path calculated by the A star algorithm;
  + The grey, transparent square represents the AI vision field;
* Press “I” to execute one iteration of the K-means algorithm. The points, if displayed on the screen, will change color according to the cluster they have been assigned to and the means will move to the centroids of the corresponding clusters.
* Press “K” to start running the K-means algorithm, one iteration will be executed every half a second and when the algorithm stops a text with information regarding he number of iterations will appear at the bottom of the screen.
* Use the “Escape” key to close the application.

When the player enters the goal area or the AI touches the player character the game will stop and a message will appear on the screen displaying the result of the game.

**CONCLUSION**