**AI Project Report**

**Polydope**

Anirudh Sureshram – 1PE16CS020

Akash N Somasalle – 1PE17CS014

**Problem Definition**

The aim of the project is to show how Artificial intelligence is being used in the modern day games. The project uses Finite state machine AI programming.

## Introduction

Elements of artificial intelligence used in computer games have come a long way. In the beginning, the developed systems were based on sets of rules written directly in the code of the game or on the behaviour scripts interpreted by the code, with the whole thing based most commonly on the appropriate selection of importance of the random factor in the process of choosing the appropriate behaviour. That time witnessed the birth of such memorable games as the immortal River-Raid, Donkey-Kong, Boulder-Dash, and many other objects of fascination for users of eight-bit machines, back in the 1970s.

Another step in the development process was introducing simple computer science methods, such as the still popular and frequently used Finite State Machine method, into describing the behaviour of the computer-controlled enemies. However, as the demands of the players grew day by day, games grew more and more complicated, thanks to the use of more and more advanced computing algorithms. The dawn of the era of RTS-type games (Real Time Strategy) has caused a significant shift of interest (in terms of frequency of use) to algorithms which determine the optimal path between two specified points on a map.

For a long time, the foremost indicator of a computer game's quality was the quality of three-dimensional graphics it featured; however, a realisation soon came that nice graphics, sound, and character animation is not everything. Recently, one of the most important elements of computer games has been identified as artificial intelligence – as the primary factor behind the so-called playability of present-day video games.

The process of production of computer games has undergone significant changes as well. Even though programming the artificial intelligence of a game used to be treated slightly unfairly, and its implementation tended to be pushed to near the end of the production of the game's engine, at present, planning the modules of artificial intelligence and their co-operation with other components of the game is one of the most important elements of the planning process.

More and more frequently, at least one of the members of a programming team is designated to, full-time and ever since the beginning of the project, handle designing and programming the modules of artificial intelligence.

## AI in FPS-type Games

FPS-type games usually implement the layered structure of the artificial intelligence system. Layers located at the very bottom handle the most elementary tasks, such as determining the optimal path to the target (determined by a layer higher up in the hierarchy) or playing appropriate sequences of character animation. The higher levels are responsible for tactical reasoning and selecting the behaviour which an AI agent should assume in accordance with its present strategy.

Path-finding systems are usually based on graphs describing the world. Each vertex of a graph represents a logical location (such as a room in the building, or a fragment of the battlefield). When ordered to travel to a given point, the AI agent acquires, using the graphs, subsequent navigation points it should consecutively head towards in order to reach the specified target location. Moving between navigation points, the AI system can also use local paths which make it possible to determine an exact path between two navigation points, as well as to avoid dynamically appearing obstacles.

The animation system plays an appropriate sequence of animation at the chosen speed. It should also be able to play different animation sequences for different body parts: for example, a soldier can run and aim at the enemy, and shoot and reload the weapon while still running. Games of this kind often employ the inverted kinematics system. An IK animation system can appropriately calculate the parameters of arm positioning animation so that the hand can grab an object located on, e.g., a table or a shelf. The task of modules from higher layers is to choose the behaviour appropriate for the situation – for instance, whether the agent should patrol the area, enter combat, or run through the map in search of an opponent.

Once the AI system has decided which behaviour is the most appropriate for the given situation, a lower-level module has to select the best tactics for fulfilling that task. Having received information that the agent should, for instance, fight, it tries to determine the approach that is the best at the moment – e.g., whether we should sneak up on the opponent, hide in a corner and wait for the opponent to present a target of itself, or perhaps just run at him, shooting blindly.

## Finite State Machines

Finite state machines are one of the least complicated, while at the same time, one of the most effective and most frequently used methods of programming artificial intelligence. For each object in a computer game, it is possible to discern a number of states it is in during its life. For example: a knight can be arming himself, patrolling, attacking, or resting after a battle; a peasant can be gathering wood, building a house, or defending himself against attacks. Depending on their states, in-game objects respond in different ways to (the finite set of) external stimuli or, should there be none, perform different activities. The finite state machine method lets us easily divide the implementation of each game object's behaviour into smaller fragments, which are easier to debug and extend. Each state possesses code responsible for the initialisation and deinitialisation of the object in that state (also often referred to as the state transition code), code executed in the game's each frame (e.g., to fulfill the needs of artificial intelligence functions, or to set an appropriate frame of animation), and code for processing and interpreting messages coming from the environment.

Finite state machines are typically implemented using one of the two following methods:

* finite state machine language – implemented in C as a set of preprocessor macros,
* state design pattern – a special object-oriented project pattern.

## Artificial Neural Networks and Advanced Algorithms in Computer Games

## IMG_263

The multi-layered perceptron model

Artificial neural networks could, in theory, be applied to solving most tasks performed by AI in computer games. Unfortunately, in practice, a number of obstacles exist which limit the neural networks' application in games. These include:

* problems with choosing the appropriate input for neural networks,
* neural networks' sensitivity to changes in a game's action logic, and the need for re-training the network whenever such a situation occurs,
* rather complicated theory, and difficulties with debugging in case of problems,
* time-consuming and complicated process of training the network.

What steps do we need to undertake in order to take advantage of an artificial neural network in a simple computer game? Let us have a brief look:

To begin, we have to answer our own question about the kinds of information the neural network should provide us with in order to help us solve the given problem. For example, let us consider a game in which a neural network controls the flight of our opponent's fighter plane. The information we should be obtaining from the neural network would then be, e.g., the optimal vectors of velocity and acceleration which, when provided to the physics module, will guide the enemy fighter to our plane. Another example could be a neural network used to choose the best strategy in a RTS-type game. Based on situation analysis, the network decides how greatly to concentrate on development, arms production, repairs after battles etc. All the parameters required by the game will be provided by the neural network on its output.

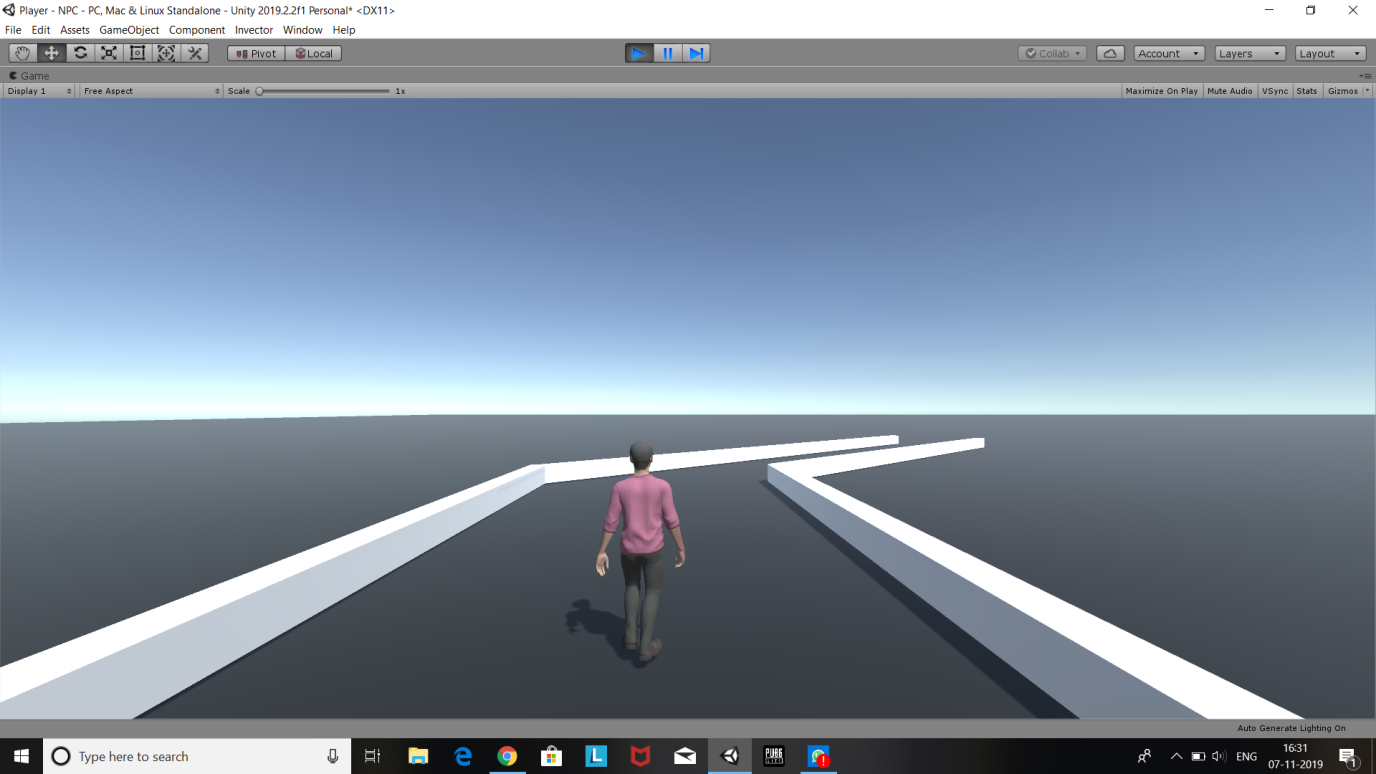
While defining the effect of the neural network's actions is quite easy (since we know exactly what we want to achieve), choosing the network's input parameters is a much more serious problem. The parameters should be chosen in such a way that its different combinations will let the neural network learn to solve complicated situations which haven't appeared in the example set of signals. The general rule states that the input data (variables) should represent as much information about the game world as possible; it could be, for instance, vectors of relative positions of the nearest obstacle and the nearest opponent, the enemy's strength, or the present state of armaments and damage.

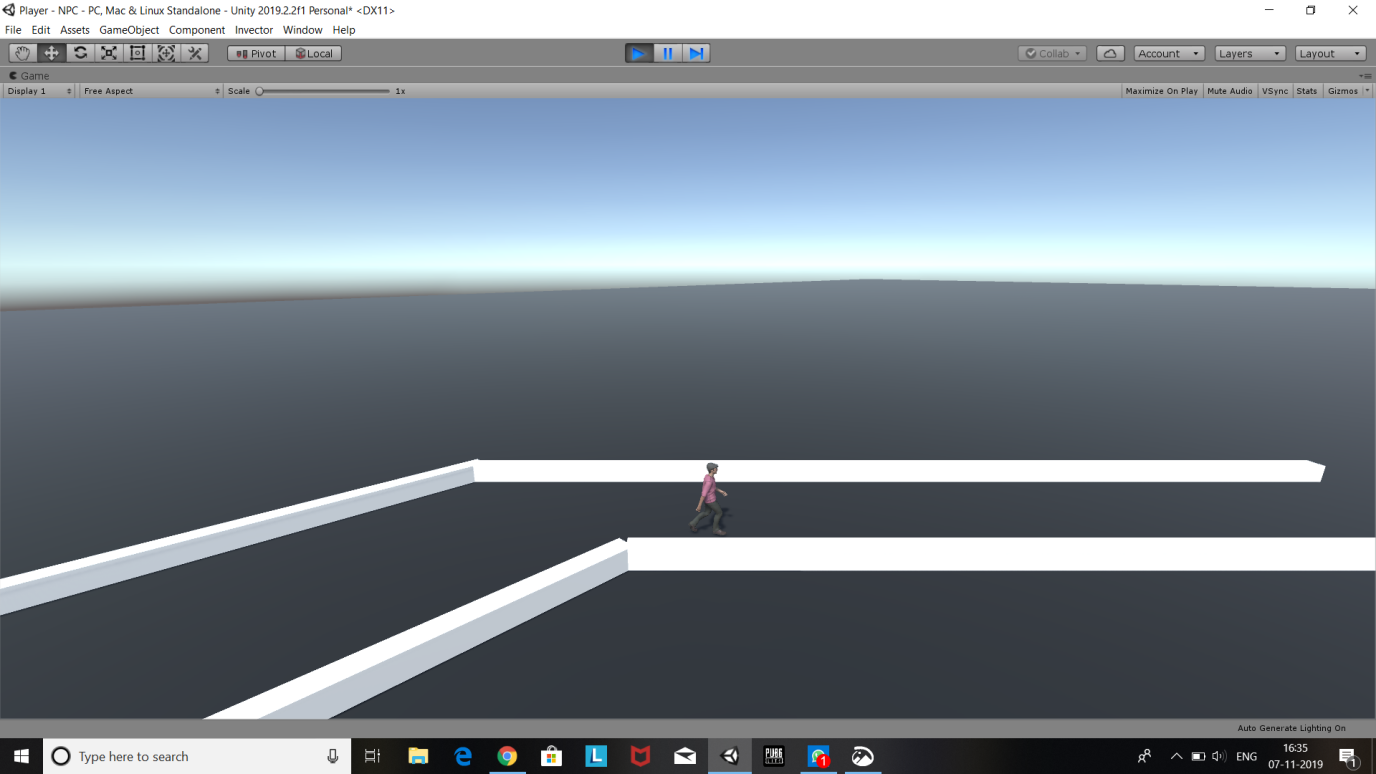
Another step is to acquire a set of input data which will be used to train the network. The direct method could imply, e.g., remembering several to several hundred samples, successful attacks, and actions of a human player, and providing the recorded data to the neural network. Typically, however, the process used is automated, i.e., the samples themselves are computer-generated – which requires an additional, often quite significant, effort from the programmers.

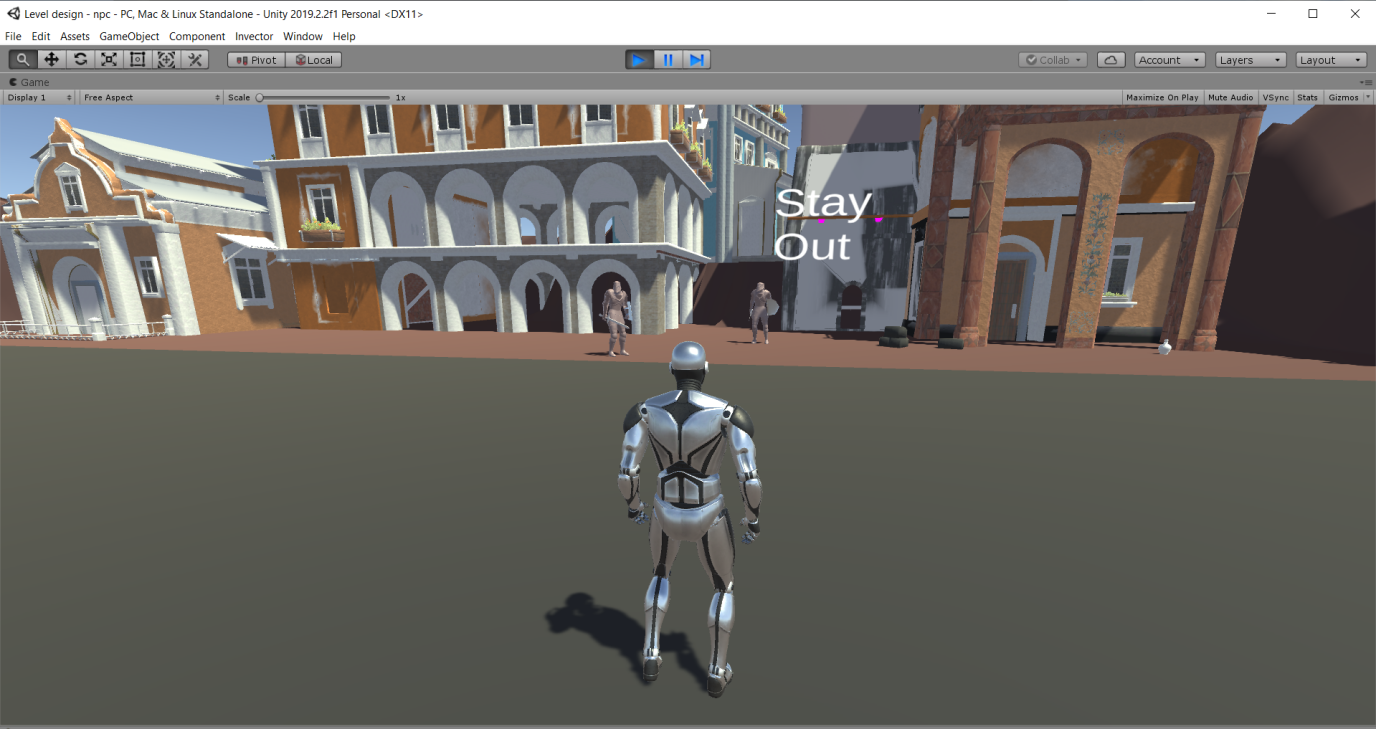
The final step is training the neural network. Any training algorithm can be used here. The training process should be interwoven with simultaneous testing in order to make sure the game is not becoming too difficult or, the opposite, if it's not still too easy and in need of further training and optimisation.

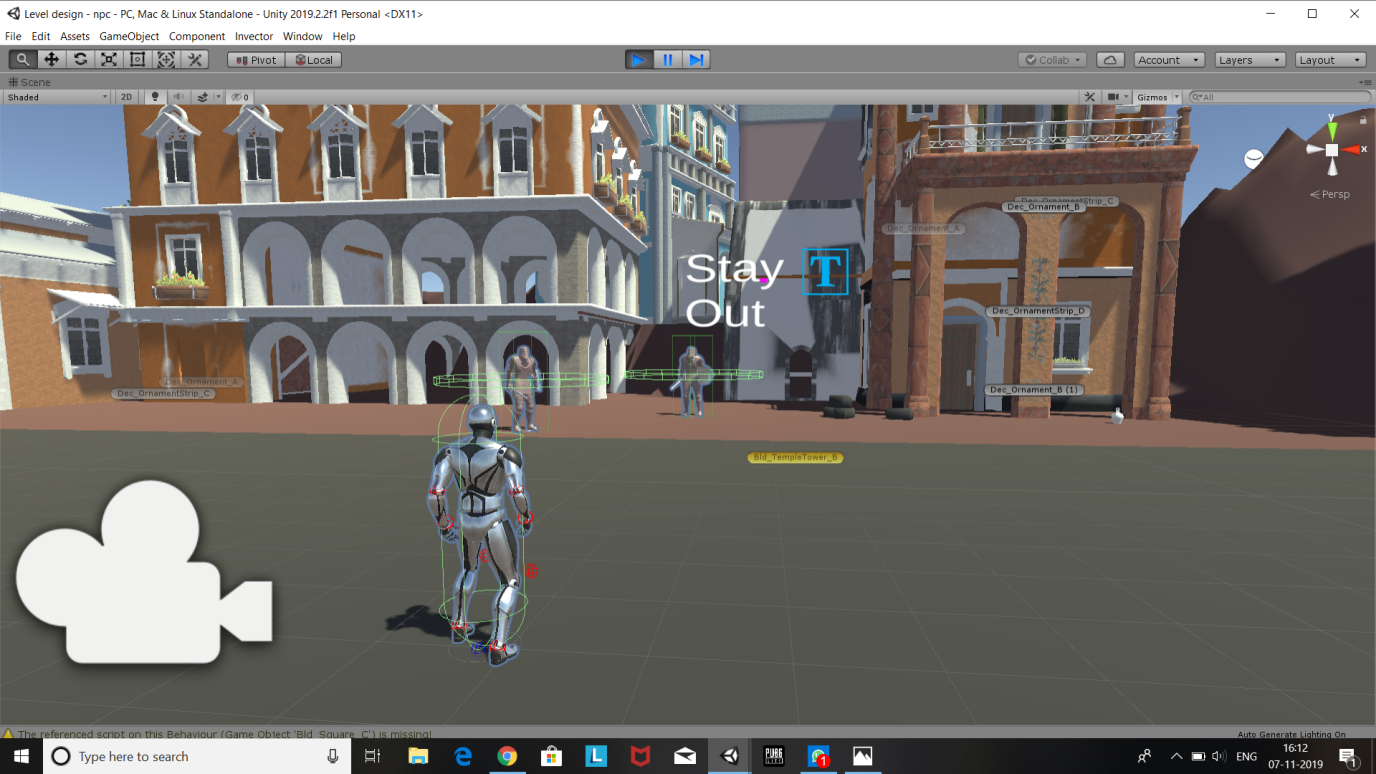
Applying neural networks practically is not an easy task. It requires a lot of time, experience, and patience. In addition, neural networks are often used together with fuzzy logic, which makes it possible to convert the computer's traditional zero-one reasoning into something more strongly resembling the way a human thinks. Logic lets us decide if and to what degree the given statement is true. Although simultaneous use of the two technologies is a difficult task, when it is successful, the results are simply breath-taking, and incomparable with what we can achieve by using rules hard-coded into the code with algorithms and traditional logic. Technologies such as neural networks, genetic algorithms, and fuzzy logic are the future of computer games – and a future that is not that distant any more.

## Output

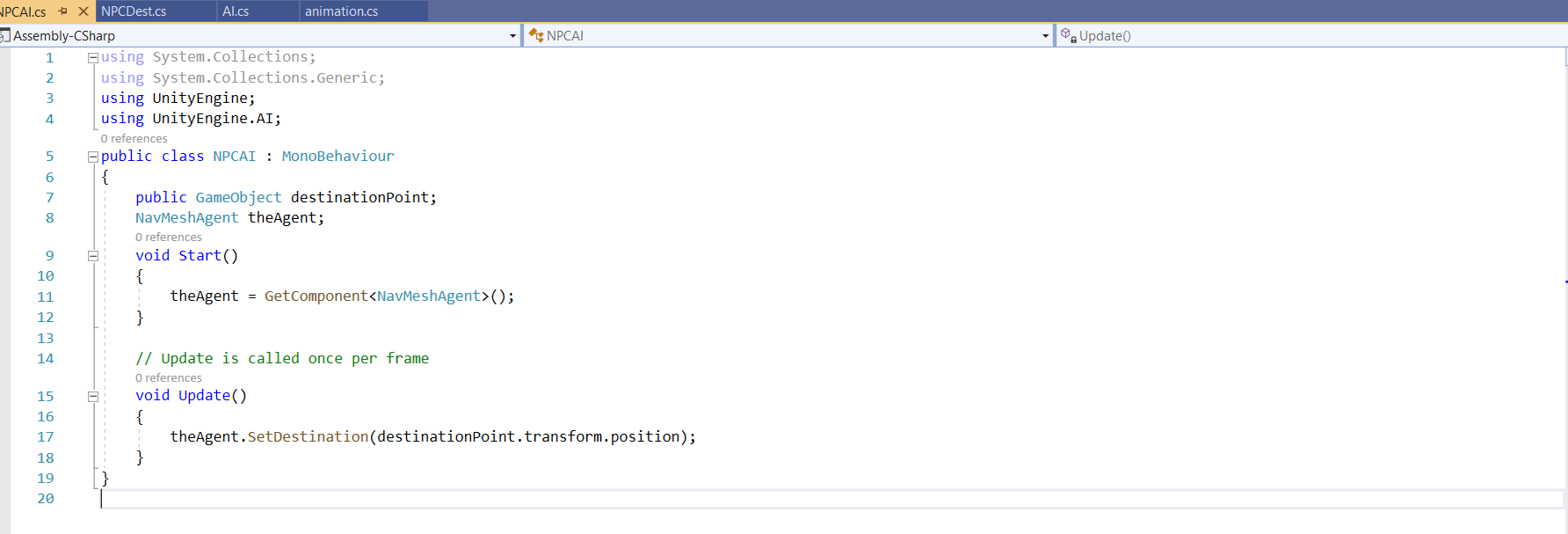


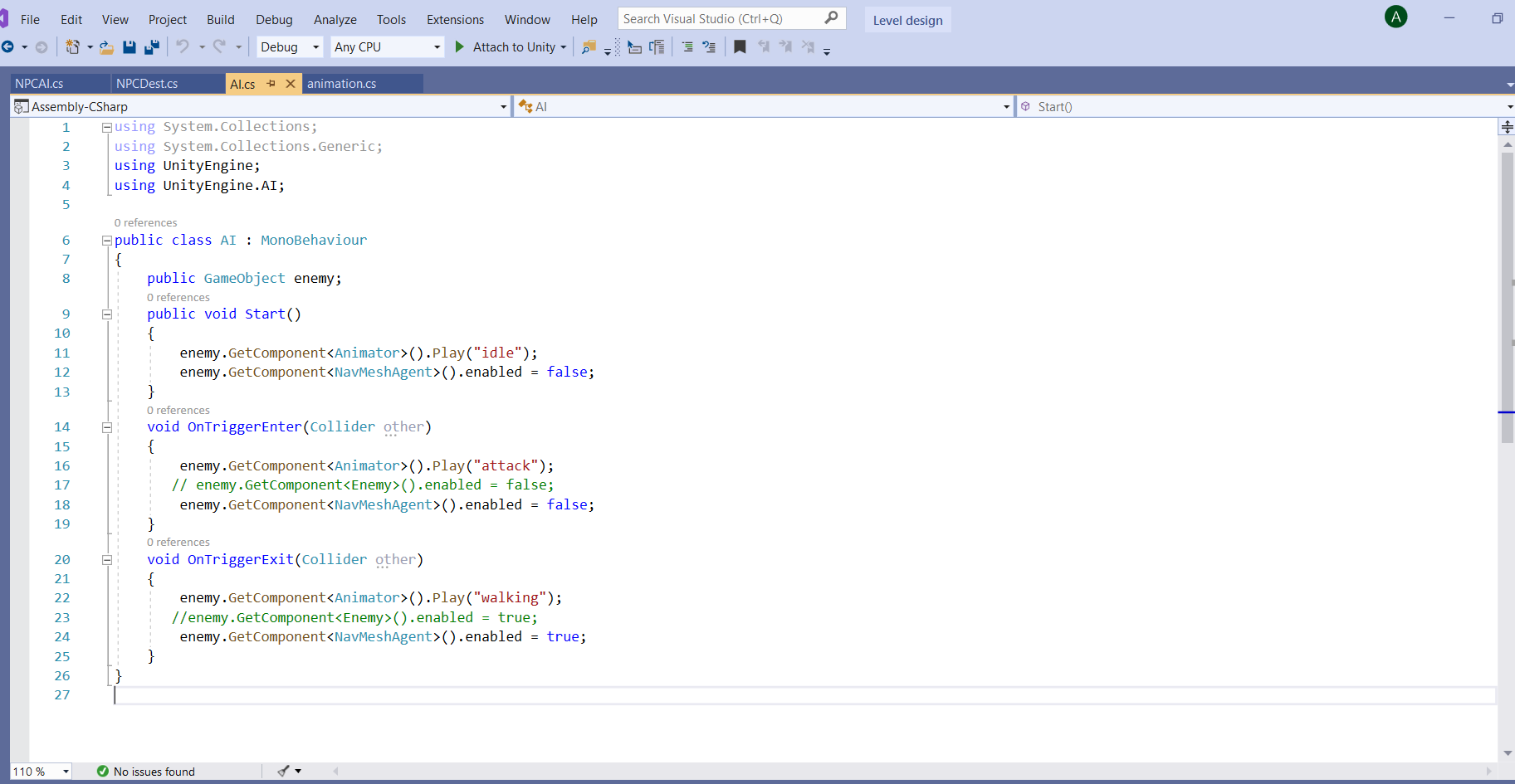


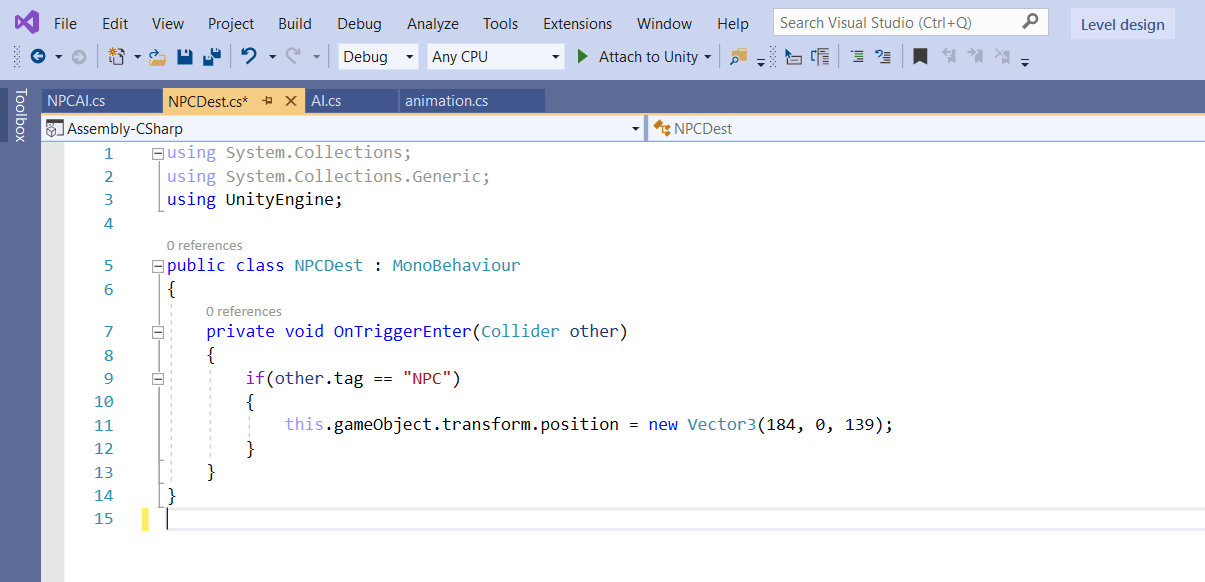




**CodeSnapshots**

****

****

****

## Conclusion

Artificial intelligence is a very broad and, at the same time, fascinating part of computer science. We now would like to increase our knowledge of the field of artificial intelligence in computer games a little more.

## What Now?

As of now, we could only implement a finite state machine AI programming.

We are very interested in using neural networks in our project that is to train the enemy and the pedestrian to react better depending on the environment.